

Gestión y Análisis de datos

Tabla de contenidos

- 1 Gestión y Análisis de datos
- 2 Tabla de contenidos
- 3 Presentación del objeto de estudio
 - 3.1 Contexto de los Datasets
 - 3.2 Conjuntos de Datos Inicial (Datasets)
 - 3.3 Fuentes de la información
 - 3.4 Fuentes de consulta
 - 3.5 Herramientas utilizadas
- 4 Motivación del objeto de estudio
 - 4.0.1 Planteo
 - 4.0.1.1 Cuestiones e Hipótesis iniciales:
- 5 Carga de datos
 - 5.1 Fuentes de los datos crudos utilizados
 - 5.2 Importar librerías
 - 5.3 Carga archivos CSV en sus respectivos dataframes
 - 5.4 Verificar carga de archivos
- 6 Limpieza de datos
 - 6.1 Verificar contenido archivo de aerolineas
 - 6.2 Verificar contenido archivo de aeropuertos
 - 6.3 Verificar contenido archivo de vuelos
 - 6.3.1 Crear DF flights_df_no_cancelados
 - 6.4 Verificar contenido archivo de US Gateway Airport(IATA_CODE) vs time zone
 - 6.5 Verificar contenido archivo de US Gateway AirportId vs US Gateway Airport(IATA_CODE)
 - 6.6 Verificar contenido archivo States
- 7 Transformación de datos
 - 7.1 Comprobación tipo de dato "mixto" para las columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT
 - 7.2 Comprobación de códigos en columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT ("IATA" vs "id aeropuerto")
 - 7.3 Se reserva los códigos IATA faltantes de ORIGIN_AIRPORT
 - 7.4 Se reserva los códigos IATA faltantes de DESTINATION_AIRPORT
 - 7.5 Corrección de los códigos IATA para ORIGIN_AIRPORT
 - 7.6 Corrección de los códigos IATA para DESTINATION_AIRPORT
 - 7.7 Se incorpora FECHA
 - 7.8 Se incorpora el nombre de la Aerolínea
 - 7.9 Se incorpora el nombre del aeropuerto
 - 7.10 Crear DF flights_retrasados
- 8 Análisis
 - 8.1 Bureau of Transportation Statistics (Oficina de Estadísticas de Transporte)

- 8.2 Análisis de columnas de tiempo
 - 8.2.1 Se comprueba horarios de origen y destino
- 8.3 Se comprueba la unidad de distancia
- 8.4 Gráfica de Tiempos
- 8.5 Se verifica País
- 8.6 Análisis de posible clave primaria
- 8.7 Se verifica tiempo de vuelo máximo en todo el dataset
- 8.8 Se verifica condición de ARRIVAL_DELAY
- 8.9 Analizar umbral a partir del cual computar una causa de retardo
- 9 Visualizaciones
 - 9.1 ¿Qué datos deberían tenerse en cuenta antes de contratar un vuelo?
 - 9.2 Análisis Mensual
 - 9.3 Análisis diario (día del mes)
 - 9.4 Análisis diario (día de la semana)
 - 9.5 Análisis horario
 - 9.6 Analizar la cantidad de vuelos por Aerolínea
 - 9.7 Análisis de aeropuertos de origen
 - 9.8 Análisis (Hip.1) e (Hip.2)
 - 9.9 Análisis (Hip.3)
 - 9.9.1 Promedios
 - 9.9.2 Medidas de Dispersión
- 10 Almacenamiento de resultados

Presentación del objeto de estudio

- [Volver](#)

Contexto de los Datasets

La Oficina de Estadísticas de Transporte del Departamento de Transporte de USA (DOT, por sus siglas en inglés) realiza un seguimiento de la puntualidad de los vuelos nacionales operados por grandes compañías aéreas. La información resumida sobre la cantidad de vuelos a tiempo, retrasados, cancelados y desviados se publica en el Informe mensual de consumidores de viajes aéreos del DOT y en este conjunto de datos de retrasos y cancelaciones de vuelos de 2015.

Conjuntos de Datos Inicial (Datasets)

- airlines.csv (359 B): Códigos y nombres de aerolíneas IATA
- airports.csv (23.8 KB): Códigos IATA y datos de aeropuertos en USA
- flights.csv (592.4 MB): Información de los vuelos 2015 en USA
- IATA_TZ_data.csv (179KB): Mapeo US Gateway Airport(IATA_CODE) Code VS Time zone Airport
- International_Report_Passengers_2015.csv (9 KB): Mapeo US Gateway Airport ID VS US Gateway Airport Code(IATA_CODE) del año 2015.

- States.csv (816 B): Estados de los EEUU

Fuentes de la información

- <https://www.bts.gov/>
- <https://datahub.io/core/airport-codes/>
- <https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/>
- <https://www.kaggle.com/datasets/usdot/flight-delays?select=flights.csv>
- <http://www.fresse.org/dateutils/tzmaps.html> LINK:
<https://raw.github.com/hroptatyr/dateutils/tzmaps/iata.tzmap>
- <https://opendatanetwork.herokuapp.com/dataset/datahub.transportation.gov/xgub-n9bw>
- <https://www.ups.com/worldshiphelp/WSA/ESP/AppHelp/mergedProjects/CORE/Codes/Stat>

Fuentes de consulta

- https://es.wikipedia.org/wiki/C%C3%B3digo_de_aeropuertos_de_IATA
- <https://www.bts.gov/topics/airlines-and-airports/understanding-reporting-causes-flight-delays-and-cancellations>
- Buscador Google:
 - What does taxi out mean in aviation?
 - What does Wheels Out mean in aviation?
- <https://es.trip.com/flights/airport-iah-rdu/george-bush-intercontinental-airport-to-raleigh-international-airport/>
- https://www.google.com/search?q=millas+a+km+distancia&rlz=1C1ASUM_enAR827AR828&oq=millas+a+km&aqs=chrome..8
- <https://www.vercalendario.info/es/como/convertir-latitud-longitud-grados-decimales.html>
- <https://www.distanc esto.com/>
- https://es.wikipedia.org/wiki/Aeropuerto_Internacional_de_Austin-Bergstrom

Herramientas utilizadas

A continuación se detallan las herramientas utilizadas para el desarrollo.

- Google Colab / Jupyter Notebook
- Paint

Motivación del objeto de estudio

- [Volver](#)

Planteo

Luego de hallado el dataset, analizado sus registros, columnas, verificada su procedencia y haciendo un análisis preliminar de las relaciones del contenido, se plantearon cuestiones e hipótesis que sirvan como puntapié inicial para el desarrollo del trabajo.

Estas cuestiones e hipótesis no surgen de un análisis técnico del sistema aéreo estadounidense sino de la necesidad de conocer en profundidad (como usuarios consumidores de vuelos), las causas de por qué un vuelo puede llegar a retrasarse y todas las variables que influyen en esa demora final que genera que un avión no llegue a destino según lo planificado.

Cuestiones e Hipótesis iniciales:

- o ¿Existe algún mes que se concentren mayor cantidad de vuelos?.
¿Algún mes tiene mayor cantidad de retrasos?.
¿Algún mes tiene mayor probabilidad de retraso?.
- o ¿Existe algún día de la semana que se concentren mayor cantidad de vuelos?.
¿Algún día de la semana tiene mayor cantidad de retrasos?.
- o ¿Existe algún día del mes que se concentren mayor cantidad de vuelos?.
¿Algún día del mes tiene mayor cantidad de retrasos?.
- o ¿Existe alguna hora que se concentren mayor cantidad de vuelos?.
¿Alguna hora tiene mayor cantidad de retrasos?.
- o ¿Existe alguna aerolínea en particular que registra mayor cantidad de vuelos demorados?.
¿Cuál es la proporción en función a los vuelos totales?.
- o ¿En qué aeropuertos de origen se da la mayor cantidad de vuelos con demora por cuestiones de seguridad?.
¿En qué aeropuertos de origen y en cuáles aerolíneas se dan mayor cantidad de vuelos con demora por cuestiones de seguridad?.
- o ¿En qué aeropuertos de origen y en cuáles aerolíneas tiene vuelos de mayor demora por cuestiones de seguridad?.
- o ¿Cuál es el umbral a partir del cual se comienza a computar una causa de retardo?.
- o (Hip.1) Las demoras en la mayoría de los vuelos se dan a causa de factores climáticos.
- o (Hip.2) La disminución de la cantidad y tiempo de demora en los vuelos puede lograrse si se disminuyen los controles de seguridad en los aeropuertos.
- o (Hip.3) El tiempo de demora de salida y llegada de los vuelos son, en su mayoría, positivos y altos.

Carga de datos

- [Volver](#)

Con el objetivo de hacer un eficiente uso de los recursos y, debido a la gran cantidad de datos procesados en el presente estudio, se estarán realizando reinicios de índice de los distintos DataFrames utilizados de forma frecuente, así como otras acciones destinadas a tal fin.

Fuentes de los datos crudos utilizados

- [airlines.csv](#)
- [airports.csv](#)
- [flights.csv](#)

Fuente: <https://www.kaggle.com/datasets/usdot/flight-delays?select=flights.csv>

- [IATA_TZ_data.csv](#)

Fuente: <http://www.fresse.org/dateutils/tzmaps.html>

link: <https://raw.github.com/hroptatyr/dateutils/tzmaps/iata.tzmap>

Nota: A este archivo se le agregó los nombres a las columnas, se limpiaron líneas en blanco y se cambió el tabulador por comas. Estas modificaciones son realizadas previo a la carga del mismo.

- [International_Report_Passengers_2015.csv](#)

Fuente: <https://opendatanetwork.herokuapp.com/dataset/datahub.transportation.gov/xgub-n9bw>

Nota: Este archivo tiene información de tráfico comercial de pasajeros sin escalas que viaja entre puntos internacionales y aeropuertos de EE. UU., se ha extraído solo las columnas necesarias(usg_apt_id, usg_apt) para el año 2015, también se han quitado duplicados. Este recorte es previo a la carga para poder manipularlo y tener condiciones óptimas de memoria y tiempos de procesamiento en el Colab / Jupyter.

- [States.csv](#)

Fuente:

https://www.ups.com/worldshiphelp/WSA/ESP/AppHelp/mergedProjects/CORE/Codes/State_Pro

Nota: A este archivo se le agregó los nombres a las columnas y se cambió el tabulador por comas.

Estas modificaciones son realizadas previo a la carga del mismo.

Advertencia:

se recomienda correr el notebook en Jupyter, por tema de espacio en memoria y velocidad de proceso.

En google colab habrá que dejar de lado algunas ejecuciones para que la memoria del mismo alcance en todo el proceso.

```
In [1]: #Ambiente de ejecución  
#Si Colab(True), Anaconda(False)  
colab=False
```

```
In [2]: if colab:  
    #Montar Google Drive  
    from google.colab import drive  
    drive.mount('/content/drive/')
```

Importar librerías

- [Volver](#)

```
In [3]: #Tratamiento de datos  
import numpy as np  
import pandas as pd  
from datetime import datetime, timedelta  
from dateutil import parser  
import datetime  
  
#Visualización  
import matplotlib.pyplot as plt  
import matplotlib.patches as mpatches  
import seaborn as sns  
  
#Mapa  
import branca.colormap as cm  
import folium  
  
#Análisis estadístico  
from scipy import stats  
  
#Creación de filtros  
from ipywidgets import interactive, interactive_output  
import ipywidgets as widgets  
from IPython.display import display  
  
#Se setea el entorno para ver todas Las columnas  
pd.set_option('display.max_columns', None)
```

Carga archivos CSV en sus respectivos dataframes

- [Volver](#)

```
In [4]: if colab:
    data_path='/content/drive/MyDrive/DataScience/'
else:
    # Jupyter: colocar en el mismo directorio del notebook (O elegir la ruta que se
    data_path='./'

data_to_check='data_to_check/'
data_result='data_result/'

airlines_path = data_path + "airlines.csv"
airport_path = data_path + "airports.csv"
flights_path = data_path + "flights.csv"
timezone_path= data_path + "IATA_TZ_data.csv"
usg_apt_path= data_path + "International_Report_Passengers_2015.csv"
states_path= data_path + "States.csv"

# Cargar datos y forzar tipos
airlines_data = pd.read_csv(airlines_path, dtype={'IATA_CODE': 'str', 'AIRLINE': 'str',
                                                 'AIRPORT': 'str', 'CARRIER': 'str',
                                                 'CARRIER_TYPE': 'str', 'CARRIER_NAME': 'str',
                                                 'DEP_TIME_BLK': 'str', 'ORIGIN_AIRPORT': 'str',
                                                 'DESTINATION_AIRPORT': 'str', 'CRS_DEP_TIME': 'str',
                                                 'CRS_ELAPSED_TIME': 'str', 'CRS_ARR_TIME': 'str',
                                                 'ARR_TIME': 'str', 'ARR_DELAY': 'str', 'ARR_DEL15': 'str',
                                                 'DEP_DELAY': 'str', 'DEP_DEL15': 'str', 'DISTANCE': 'str',
                                                 'CRS_ARR_DELAY': 'str', 'CRS_ARR15': 'str', 'CRS_DEP15': 'str',
                                                 'CRS_ELAPSED15': 'str', 'CRS_DISTANCE': 'str',
                                                 'CRS_ARR_DISTANCE': 'str'}, encoding='latin1')
airport_data = pd.read_csv(airport_path, dtype={'IATA_CODE': 'str', 'AIRPORT': 'str',
                                                'CITY': 'str', 'STATE': 'str', 'COUNTRY': 'str',
                                                'LATITUDE': 'str', 'LONGITUDE': 'str', 'TIMEZONE': 'str',
                                                'ELEVATION': 'str', 'TYPE': 'str'}, encoding='latin1')
flights_data = pd.read_csv(flights_path)
IATA_TZ_data = pd.read_csv(timezone_path, dtype={'iata_code': 'str', 'time_zone': 'str'})
usg_apt_data = pd.read_csv(usg_apt_path, dtype={'usg_apt_id': 'str', 'usg_apt': 'str'})
states_data = pd.read_csv(states_path, dtype={'STATE': 'str', 'CODE': 'str'})
```

```
C:\Users\Public\Documents\Wondershare\CreatorTemp\ipykernel_176212\3292390776.py:2
0: DtypeWarning: Columns (7,8) have mixed types. Specify dtype option on import or
set low_memory=False.
    flights_data = pd.read_csv(flights_path)
```

Nota: Se revisará luego el Warning que se está presentando: "Columns (7,8) have mixed types". Se trata de las columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT del DataFrame Flights.

```
In [5]: # Necesarios para ipywidgets.widgets
months=[('All',0),('Enero',1),('Febrero',2),('Marzo',3),('Abril',4),('Mayo',5),('Junio',6),
       ('Julio',7),('Agosto',8),('Septiembre',9),('Octubre',10),('Noviembre',11),('Diciembre',12)]

# Necesarios para el título en las funciones de visualización
months_dic ={0:'All',1:'Enero',2:'Febrero',3:'Marzo',4:'Abril',5:'Mayo',6:'Junio',
             7:'Julio',8:'Agosto',9:'Septiembre',10:'Octubre',11:'Noviembre',12:'Diciembre'}
```

Verificar carga de archivos

- [Volver](#)

- Verificar archivo de aerolineas

```
In [6]: airlines_data.head()
```

Out[6]:

	IATA_CODE	AIRLINE
0	UA	United Air Lines Inc.
1	AA	American Airlines Inc.
2	US	US Airways Inc.
3	F9	Frontier Airlines Inc.
4	B6	JetBlue Airways

- Verificar archivo de aeropuertos

In [7]: `airport_data.head()`

Out[7]:

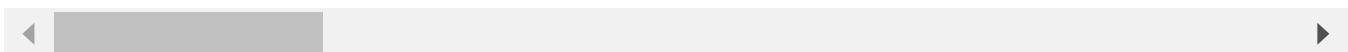
	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

- Verificar archivo de vuelos

In [8]: `flights_data.head()`

Out[8]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPC
0	2015	1	1	4	AS	98	N407AS	A
1	2015	1	1	4	AA	2336	N3KUAA	I
2	2015	1	1	4	US	840	N171US	S
3	2015	1	1	4	AA	258	N3HYAA	I
4	2015	1	1	4	AS	135	N527AS	S



- Verificar archivo de US Gateway Airport(IATA_CODE) vs time zone

In [9]: `IATA_TZ_data.head()`

Out[9]:

	iata_code	time_zone
0	AAA	Pacific/Tahiti
1	AAB	Australia/Brisbane
2	AAC	Africa/Cairo
3	AAD	Africa/Mogadishu
4	AAE	Africa/Algiers

- Verificar archivo de US Gateway AirportId vs US Gateway Airport(IATA_CODE)

In [10]: `usg_apt_data.head()`

Out[10]:

	usg_apt_id	usg_apt
0	10010	1B1
1	10135	ABE
2	10136	ABI
3	10140	ABQ
4	10141	ABR

- Verificar archivo States

In [11]: `states_data.head()`

Out[11]:

	STATE	CODE
0	Alabama	AL
1	Alaska	AK
2	Arizona	AZ
3	Arkansas	AR
4	California	CA

Limpieza de datos

- [Volver](#)

Verificar contenido archivo de aerolineas

- [Volver](#)

In [12]: `airlines_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   IATA_CODE   14 non-null    object  
 1   AIRLINE     14 non-null    object  
dtypes: object(2)
memory usage: 352.0+ bytes
```

In [13]: `airlines_data.describe()`

Out[13]:

	IATA_CODE	AIRLINE
count	14	14
unique	14	14
top	UA	United Air Lines Inc.
freq	1	1

Verificar contenido archivo de aeropuertos

- [Volver](#)

In [14]: `airport_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   IATA_CODE   322 non-null    object  
 1   AIRPORT     322 non-null    object  
 2   CITY         322 non-null    object  
 3   STATE        322 non-null    object  
 4   COUNTRY     322 non-null    object  
 5   LATITUDE    319 non-null    float64 
 6   LONGITUDE   319 non-null    float64 
dtypes: float64(2), object(5)
memory usage: 17.7+ KB
```

Se verifica que faltan 3 datos de latitud y longitud

Como son pocos, se completa de forma manual.

- Fuente: <https://www.vercalendario.info/es/como/convertir-latitud-longitud-grados-decimales.html>
- Fuente: <https://www.distanciesto.com/>

In [15]: `#Obtener las posiciones a completar.
airport_data[airport_data['LATITUDE'].isna()]`

Out[15]:

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
96	ECP	Northwest Florida Beaches International Airport	Panama City	FL	USA	NaN	NaN
234	PBG	Plattsburgh International Airport	Plattsburgh	NY	USA	NaN	NaN
313	UST	Northeast Florida Regional Airport (St. Augustine...)	St. Augustine	FL	USA	NaN	NaN

In [16]: #Se completa la latitud y longitud

```
airport_data.iloc[96, 5] = 30.352934
airport_data.iloc[96, 6] = -85.79427
airport_data.iloc[234, 5] = 44.669441
airport_data.iloc[234, 6] = -73.472294
airport_data.iloc[313, 5] = 29.954705
airport_data.iloc[313, 6] = -81.343314
```

In [17]: airport_data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
 ---  --          --          --    
 0   IATA_CODE    322 non-null    object 
 1   AIRPORT      322 non-null    object 
 2   CITY          322 non-null    object 
 3   STATE         322 non-null    object 
 4   COUNTRY       322 non-null    object 
 5   LATITUDE      322 non-null    float64
 6   LONGITUDE     322 non-null    float64
dtypes: float64(2), object(5)
memory usage: 17.7+ KB
```

Verificar contenido archivo de vuelos

- [Volver](#)

In [18]: flights_data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5819079 entries, 0 to 5819078
Data columns (total 31 columns):
 #   Column           Dtype  
 --- 
 0   YEAR            int64  
 1   MONTH           int64  
 2   DAY             int64  
 3   DAY_OF_WEEK     int64  
 4   AIRLINE          object  
 5   FLIGHT_NUMBER   int64  
 6   TAIL_NUMBER     object  
 7   ORIGIN_AIRPORT  object  
 8   DESTINATION_AIRPORT  object  
 9   SCHEDULED_DEPARTURE  int64  
 10  DEPARTURE_TIME  float64 
 11  DEPARTURE_DELAY float64 
 12  TAXI_OUT         float64 
 13  WHEELS_OFF       float64 
 14  SCHEDULED_TIME  float64 
 15  ELAPSED_TIME    float64 
 16  AIR_TIME         float64 
 17  DISTANCE         int64  
 18  WHEELS_ON        float64 
 19  TAXI_IN          float64 
 20  SCHEDULED_ARRIVAL  int64  
 21  ARRIVAL_TIME    float64 
 22  ARRIVAL_DELAY   float64 
 23  DIVERTED         int64  
 24  CANCELLED        int64  
 25  CANCELLATION_REASON  object  
 26  AIR_SYSTEM_DELAY float64 
 27  SECURITY_DELAY   float64 
 28  AIRLINE_DELAY    float64 
 29  LATE_AIRCRAFT_DELAY  float64 
 30  WEATHER_DELAY   float64 
dtypes: float64(16), int64(10), object(5)
memory usage: 1.3+ GB
```

```
In [19]: flights_data.isnull().sum()
```

```
Out[19]:
```

YEAR	0
MONTH	0
DAY	0
DAY_OF_WEEK	0
AIRLINE	0
FLIGHT_NUMBER	0
TAIL_NUMBER	14721
ORIGIN_AIRPORT	0
DESTINATION_AIRPORT	0
SCHEDULED_DEPARTURE	0
DEPARTURE_TIME	86153
DEPARTURE_DELAY	86153
TAXI_OUT	89047
WHEELS_OFF	89047
SCHEDULED_TIME	6
ELAPSED_TIME	105071
AIR_TIME	105071
DISTANCE	0
WHEELS_ON	92513
TAXI_IN	92513
SCHEDULED_ARRIVAL	0
ARRIVAL_TIME	92513
ARRIVAL_DELAY	105071
DIVERTED	0
CANCELLED	0
CANCELLATION_REASON	5729195
AIR_SYSTEM_DELAY	4755640
SECURITY_DELAY	4755640
AIRLINE_DELAY	4755640
LATE_AIRCRAFT_DELAY	4755640
WEATHER_DELAY	4755640

dtype: int64

- Limpiar nulos

```
In [20]: flights_data['AIR_SYSTEM_DELAY'].replace(np.nan, 0, inplace = True)  
flights_data['SECURITY_DELAY'].replace(np.nan, 0, inplace = True)  
flights_data['AIRLINE_DELAY'].replace(np.nan, 0, inplace = True)  
flights_data['LATE_AIRCRAFT_DELAY'].replace(np.nan, 0, inplace = True)  
flights_data['WEATHER_DELAY'].replace(np.nan, 0, inplace = True)
```

- Se analiza los cancelados y derivados para ver si se tienen en cuenta

Nota: El análisis se ampliará en la sección de análisis sobre el conjunto correspondiente

```
In [21]: mask = (flights_data['CANCELLED'] == 1) | (flights_data['DIVERTED'] == 1)  
flights_data.loc[mask]['YEAR'].count()
```

```
Out[21]: 105071
```

```
In [22]: mask = ((flights_data['CANCELLED'] == 1) | (flights_data['DIVERTED'] == 1)) & (fli  
flights_data.loc[mask]['YEAR'].count()
```

```
Out[22]: 105071
```

```
In [23]: mask = ((flights_data["DIVERTED"] ==1 ) | (flights_data["CANCELLED"] ==1 )) & (0 ==  
flights_data.loc[mask]['YEAR'].count())
```

Out[23]: 105071

- Se observa que existen 105071 registros de vuelos cancelados y derivados
- Para los cancelados obviamente no existe información de arribo y retardos
- Para los derivados la información de arribo es siempre nula (no informada) y los retardos son ceros o nulos

Descartar vuelos derivados y cancelados del conjunto de análisis

Crear DF flights_df_no_cancelados

- [Volver](#)

```
In [24]: mask = (flights_data['CANCELLED'] == 0) & (flights_data['DIVERTED'] == 0)
flights_df_no_cancelados = flights_data.loc[mask].copy(deep=True)
```

```
In [25]: flights_df_no_cancelados.head()
```

Out[25]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPC
0	2015	1	1	4	AS	98	N407AS	A
1	2015	1	1	4	AA	2336	N3KUAA	I
2	2015	1	1	4	US	840	N171US	S
3	2015	1	1	4	AA	258	N3HYAA	I
4	2015	1	1	4	AS	135	N527AS	S

```
In [26]: flights_df_no_cancelados = flights_df_no_cancelados.drop(columns=['DIVERTED', 'CANC
```

```
In [27]: flights_df_no_cancelados.isnull().sum()
```

```
Out[27]: YEAR          0  
MONTH         0  
DAY           0  
DAY_OF_WEEK   0  
AIRLINE        0  
FLIGHT_NUMBER 0  
TAIL_NUMBER   0  
ORIGIN_AIRPORT 0  
DESTINATION_AIRPORT 0  
SCHEDULED_DEPARTURE 0  
DEPARTURE_TIME 0  
DEPARTURE_DELAY 0  
TAXI_OUT       0  
WHEELS_OFF     0  
SCHEDULED_TIME 0  
ELAPSED_TIME   0  
AIR_TIME        0  
DISTANCE        0  
WHEELS_ON       0  
TAXI_IN         0  
SCHEDULED_ARRIVAL 0  
ARRIVAL_TIME   0  
ARRIVAL_DELAY   0  
AIR_SYSTEM_DELAY 0  
SECURITY_DELAY   0  
AIRLINE_DELAY    0  
LATE_AIRCRAFT_DELAY 0  
WEATHER_DELAY    0  
dtype: int64
```

Verificar contenido archivo de US Gateway Airport(IATA_CODE) vs time zone

- [Volver](#)

```
In [28]: IATA_TZ_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 9054 entries, 0 to 9053  
Data columns (total 2 columns):  
 #   Column      Non-Null Count  Dtype    
 ---  --          --          --          --  
 0   iata_code   9054 non-null   object   
 1   time_zone   9054 non-null   object   
dtypes: object(2)  
memory usage: 141.6+ KB
```

```
In [29]: IATA_TZ_data.describe()
```

```
Out[29]:      iata_code      time_zone  
count      9054          9054  
unique     9054          382  
top        AAA  America/Chicago  
freq        1              609
```

Verificar contenido archivo de US Gateway AirportId vs US Gateway Airport(IATA_CODE)

- [Volver](#)

In [30]: `usg_apt_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 858 entries, 0 to 857
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  --          --          --      
 0   usg_apt_id  858 non-null    object 
 1   usg_apt     858 non-null    object 
dtypes: object(2)
memory usage: 13.5+ KB
```

In [31]: `usg_apt_data.describe()`

Out[31]:

	usg_apt_id	usg_apt
count	858	858
unique	858	858
top	10010	1B1
freq	1	1

Verificar contenido archivo States

- [Volver](#)

In [32]: `states_data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54 entries, 0 to 53
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype  
---  --       --          --      
 0   STATE    54 non-null    object 
 1   CODE     54 non-null    object 
dtypes: object(2)
memory usage: 992.0+ bytes
```

In [33]: `states_data.describe()`

Out[33]:

	STATE	CODE
count	54	54
unique	54	54
top	Alabama	AL
freq	1	1

Transformación de datos

- [Volver](#)

Comprobación tipo de dato "mixto" para las columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT

Se analiza los tipos de datos de las columnas para obtener mayor información sobre el warning presentado en la columna de aeropuerto, durante la carga del archivo.

```
In [34]: for column in flights_df_no_cancelados.columns:  
    print(f'{column}: {pd.api.types.infer_dtype(flights_df_no_cancelados[column])}')
```

YEAR: integer
MONTH: integer
DAY: integer
DAY_OF_WEEK: integer
AIRLINE: string
FLIGHT_NUMBER: integer
TAIL_NUMBER: string
ORIGIN_AIRPORT: mixed-integer
DESTINATION_AIRPORT: mixed-integer
SCHEDULED_DEPARTURE: integer
DEPARTURE_TIME: floating
DEPARTURE_DELAY: floating
TAXI_OUT: floating
WHEELS_OFF: floating
SCHEDULED_TIME: floating
ELAPSED_TIME: floating
AIR_TIME: floating
DISTANCE: integer
WHEELS_ON: floating
TAXI_IN: floating
SCHEDULED_ARRIVAL: integer
ARRIVAL_TIME: floating
ARRIVAL_DELAY: floating
AIR_SYSTEM_DELAY: floating
SECURITY_DELAY: floating
AIRLINE_DELAY: floating
LATE_AIRCRAFT_DELAY: floating
WEATHER_DELAY: floating

Se comprueba el tipo de dato "mixto" para la columna ORIGIN_AIRPORT y DESTINATION_AIRPORT, aunque dtype de Python está indicando que es object.

```
In [35]: flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len().value_counts()
```

```
Out[35]: 3.0    5231130  
5.0     26881  
Name: ORIGIN_AIRPORT, dtype: int64
```

```
In [36]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3)  
sorted(flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique())[0]
```

```
Out[36]: '10135'
```

```
In [37]: flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT'] == 10135].head
```

Out[37]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
4392604	2015	10	1	4	EV	5033	N867AS	
4392712	2015	10	1	4	EV	6011	N12996	
4393536	2015	10	1	4	EV	5175	N851AS	
4396622	2015	10	1	4	EV	5317	N858AS	
4397827	2015	10	1	4	EV	5145	N849AS	

In [38]:

```
mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3)
cont = 0

for val in sorted(flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique()):
    cont+=1
    print(f"Procesando: {cont}")

    num1=flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']==val]
    if (type(val) == type(0)):
        num2=flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']==val].index[0]
        print(f"{val} : ", f"NUM: {num1}", f"STR: {num2}")
        break
    else:
        num2=flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']==val].index[0]
        print(f"{val} : ", f"NUM: {num2}", f"STR: {num1}")
        break
```

Procesando: 1

10135 : NUM: 226 STR: 13

Se concluye que no solo se mezclo el código de IATA con el ID de aeropuerto, sino que dentro del Id de aeropuerto entraron algunos como string y otros como integer.

Nota: Se obtuvo solo el primer registro, para tener una muestra de análisis.

In [39]: flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']==10135].index[0]

Out[39]: 226

In [40]: flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']=='10135'].index[0]

Out[40]: 13

Se comprueba el tipo de dato "mixto" para la columna DESTINATION_AIRPORT, aunque Python está indicando object.

In [41]: flights_df_no_cancelados[flights_df_no_cancelados['DESTINATION_AIRPORT']==10135].index[0]

Out[41]: 224

In [42]: flights_df_no_cancelados[flights_df_no_cancelados['DESTINATION_AIRPORT']=='10135']

Out[42]: 11

Se fuerzan los tipos de datos que se consideran adecuados para cada una de las columnas.

```
In [43]: flights_df_no_cancelados[['AIRLINE', 'TAIL_NUMBER', 'ORIGIN_AIRPORT', 'DESTINATION_AIRPORT']] = flights_df_no_cancelados[['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'FLIGHT_NUMBER', 'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'TAXI_OUT', 'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME', 'ARRIVAL_DELAY', 'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY']] = flights_df_no_cancelados[['YEAR', 'MONTH', 'DAY', 'DAY_OF_WEEK', 'FLIGHT_NUMBER', 'SCHEDULED_DEPARTURE', 'DEPARTURE_TIME', 'TAXI_OUT', 'WHEELS_OFF', 'SCHEDULED_TIME', 'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'SCHEDULED_ARRIVAL', 'ARRIVAL_TIME', 'ARRIVAL_DELAY', 'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY']].astype(int)
```

Se comprueba que se haya corregido el warning de dato "mixto".

```
In [44]: flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']==10135].index.size
```

Out[44]: 0

```
In [45]: flights_df_no_cancelados[flights_df_no_cancelados['ORIGIN_AIRPORT']=='10135'].index.size
```

Out[45]: 239

```
In [46]: flights_df_no_cancelados[flights_df_no_cancelados['DESTINATION_AIRPORT']==10135].index.size
```

Out[46]: 0

```
In [47]: flights_df_no_cancelados[flights_df_no_cancelados['DESTINATION_AIRPORT']=='10135'].index.size
```

Out[47]: 235

Comprobación de códigos en columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT ("IATA" vs "id aeropuerto")

- [Volver](#)
- El código IATA tiene una longitud de 3 caracteres según:
https://es.wikipedia.org/wiki/C%C3%B3digo_de_aeropuertos_de_IATA

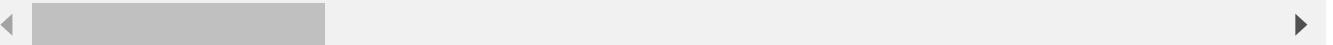
```
In [48]: # Existen con longitud mayor a 3 (algo está mal o se usó otra codificación)
mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3) | (flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3)
flights_df_no_cancelados[mask].index.size
```

Out[48]: 482878

```
In [49]: flights_df_no_cancelados[mask].head()
```

Out[49]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
4385712	2015	10	1	4	AA	1230	N3DBAA	
4385713	2015	10	1	4	DL	1805	N696DL	
4385714	2015	10	1	4	NK	612	N602NK	
4385715	2015	10	1	4	AA	260	N3GNAA	
4385716	2015	10	1	4	AA	1982	N914UY	



- Se verifica que para algunos registros las columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT no tienen el código IATA sino el id de aeropuerto

In [50]:

```
mask=(flights_df_no_cancelados['MONTH']!=10) & ((flights_df_no_cancelados['ORIGIN_'
    (flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3))
flights_df_no_cancelados[mask].index.size
```

Out[50]:

```
0
```

- Cantidad de registros con código erróneo

In [51]:

```
mask=(flights_df_no_cancelados['MONTH']==10) & ((flights_df_no_cancelados['ORIGIN_'
    (flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3))
flights_df_no_cancelados[mask].index.size
```

Out[51]:

```
482878
```

- Como información adicional se verifica que solo se dió en el mes de octubre.
De los 5.714.008 registros solo 482.878 tienen este dato con id de aeropuerto.

- Columnas ORIGIN_AIRPORT y DESTINATION_AIRPORT

In [52]:

```
#Porcentaje de error de código antes de La transformación
mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3)
round((flights_df_no_cancelados[mask].index.size/flights_df_no_cancelados.index.size)*100)
```

Out[52]:

```
8.45
```

In [53]:

```
#Porcentaje de error de código antes de La transformación
mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3)
round((flights_df_no_cancelados[mask].index.size/flights_df_no_cancelados.index.size)*100)
```

Out[53]:

```
8.45
```

Adicionar a Dataframe flights_df_no_cancelados la información del Dataframe usg_apt_data con la traducción de los códigos necesarios para corregir.

In [54]:

```
flights_df_no_cancelados=pd.merge(left=flights_df_no_cancelados, right=usg_apt_data)
```

In [55]:

```
flights_df_no_cancelados=pd.merge(left=flights_df_no_cancelados, right=usg_apt_data)
```

Se verifica el contenido luego del merge.

```
In [56]: mask=(flights_df_no_cancelados['ORIGIN_ID'].notna()) | (flights_df_no_cancelados['I' + flights_df_no_cancelados['mask']]
```

Out[56]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
4299046	2015	10	1	4	AA	1230	N3DBAA	
4299047	2015	10	1	4	DL	1805	N696DL	
4299048	2015	10	1	4	NK	612	N602NK	
4299049	2015	10	1	4	AA	260	N3GNAA	
4299050	2015	10	1	4	AA	1982	N914UY	
...
4781919	2015	10	31	6	B6	98	N715JB	
4781920	2015	10	31	6	B6	839	N834JB	
4781921	2015	10	31	6	B6	1503	N907JB	
4781922	2015	10	31	6	UA	1696	N68836	
4781923	2015	10	31	6	UA	717	N69829	

482722 rows × 32 columns

```
In [57]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()<=3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.contains('I'))  
sin_codigo = flights_df_no_cancelados[mask].index.size  
  
print(f"Cantidad de registros de ORIGIN_AIRPORT sin poder arreglar: {sin_codigo}")
```

Cantidad de registros de ORIGIN_AIRPORT sin poder arreglar: 1560

Se reserva los códigos IATA faltantes de ORIGIN_AIRPORT

- [Volver](#)
- De los 1560 casos de ORIGIN_AIRPORT, se reservarán los códigos faltantes para tratamiento futuro

```
In [58]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()<=3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.contains('I'))  
s_Temp=pd.Series(sorted(flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique()))  
s_Temp.to_csv(data_path + data_to_check + 'ORIGIN_AIRPORT_IDS.csv',index=False,header=False)  
print(s_Temp.tolist())
```

['10157', '10333', '10631', '10918', '10926', '11076', '11097', '11315', '11447', '11587', '11617', '12177', '12402', '12519', '13241', '14109', '14520', '14543']

```
In [59]: #Porcentaje de error de código que quedará Luego de La transformación  
round((sin_codigo/flights_df_no_cancelados.index.size)*100,2)
```

```
Out[59]: 0.03
```

```
In [60]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['DESTINATION_AIRPORT'].str.contains('AA'))  
sin_codigo = flights_df_no_cancelados[mask].index.size  
  
print(f"Cantidad de registros de DESTINATION_AIRPORT sin poder arreglar: {sin_codigo}")  
  
Cantidad de registros de DESTINATION_AIRPORT sin poder arreglar: 1555
```

Se reserva los códigos IATA faltantes de DESTINATION_AIRPORT

- [Volver](#)
- De los 1555 casos de DESTINATION_AIRPORT, se reservarán los códigos faltantes para tratamiento futuro

```
In [61]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['DESTINATION_AIRPORT'].str.contains('AA'))  
s_Temp=pd.Series(sorted(flights_df_no_cancelados[mask]['DESTINATION_AIRPORT'].unique))  
s_Temp.to_csv(data_path + data_to_check + 'DESTINATION_AIRPORT_IDS.csv',index=False)  
  
print(s_Temp.tolist())  
  
['10157', '10333', '10631', '10918', '10926', '11076', '11097', '11315', '11447',  
'11587', '11617', '12177', '12402', '12519', '13241', '14109', '14520', '14543']
```

```
In [62]: #Porcentaje de error de código que quedará luego de la transformación  
round((sin_codigo/flights_df_no_cancelados.index.size)*100,2)
```

```
Out[62]: 0.03
```

Corrección de los códigos IATA para ORIGIN_AIRPORT

- [Volver](#)
- Se eliminan los índices obtenidos del dataframe, para los que no se pueden reemplazar

```
In [63]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.contains('AA'))  
flights_df_no_cancelados.drop(flights_df_no_cancelados[mask].index, inplace=True)
```

- Se verifica la operación

```
In [64]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3) & (flights_df_no_cancelados['ORIGIN_AIRPORT'].str.contains('AA'))  
flights_df_no_cancelados[mask].index.size
```

```
Out[64]: 0
```

Se corrigen los casos en que sí, se tiene códigos.

```
In [65]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3)
flights_df_no_cancelados.loc[mask,'ORIGIN_AIRPORT']=flights_df_no_cancelados.loc[m

```

```
In [66]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT'].str.len()>3)
flights_df_no_cancelados[mask].index.size

```

```
Out[66]: 0
```

```
In [67]: flights_df_no_cancelados.index.size
```

```
Out[67]: 5712448
```

```
In [68]: flights_df_no_cancelados = flights_df_no_cancelados.drop(columns=['ORIGIN_ID','ORI
```

Corrección de los códigos IATA para DESTINATION_AIRPORT

- [Volver](#)

- Se eliminan los índices obtenidos del dataframe, para los que no se pueden reemplazar

```
In [69]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3) & (flights_df_no_
flights_df_no_cancelados.drop(flights_df_no_cancelados[mask].index, inplace=True)
```

- Se verifica la operación

```
In [70]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3) & (flights_df_no_
flights_df_no_cancelados[mask].index.size
```

```
Out[70]: 0
```

Se corrigen los casos en que sí, se tiene códigos.

```
In [71]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3)
flights_df_no_cancelados.loc[mask,'DESTINATION_AIRPORT']=flights_df_no_cancelados.
```

```
In [72]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT'].str.len()>3)
flights_df_no_cancelados[mask].index.size
```

```
Out[72]: 0
```

```
In [73]: flights_df_no_cancelados.index.size
```

```
Out[73]: 5711049
```

```
In [74]: flights_df_no_cancelados = flights_df_no_cancelados.drop(columns=['DESTINATION_ID'])
```

Se logró disminuir el porcentaje de error, de la siguiente manera:

ORIGIN_AIRPORT: Se paso de un porcentaje con error de 8.45% a 0.03%(1560 casos).

DESTINATION_AIRPORT: Se paso de un porcentaje con error de 8.45% a 0.03%(1555 casos).

Los casos sin poder solucionar se guardaron para su posterior análisis y fueron eliminados del dataset.

Se incorpora FECHA

- [Volver](#)
- Se incorpora FECHA (YYYY-MM-DD) con la concatenación de ['YEAR', 'MONTH', 'DAY']

```
In [75]: flights_df_no_cancelados = pd.concat([flights_df_no_cancelados, pd.to_datetime(flights_df_no_cancelados['DATE'])], axis=1)
flights_df_no_cancelados.rename(columns={0:'FECHA'}, inplace=True)
flights_df_no_cancelados = flights_df_no_cancelados.sort_values('FECHA', ascending=True)
```

```
In [76]: flights_df_no_cancelados.head()
```

```
Out[76]:
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT
0	2015	1	1	4	AS	98	N407AS	
8969	2015	1	1	4	WN	1133	N627SW	
8970	2015	1	1	4	WN	1757	N752SW	
8971	2015	1	1	4	WN	1587	N8309C	
8972	2015	1	1	4	WN	4738	N8302F	

```
In [77]: flights_df_no_cancelados.index.size
```

```
Out[77]: 5711049
```

Se incorpora el nombre de la Aerolínea

- [Volver](#)
- Se crea copia por si hay que volver atrás

```
In [78]: flights_df_no_cancelados_Temp_A = pd.merge(left = flights_df_no_cancelados, right = flights_df_no_cancelados[['AIRLINE']], how='left')
```

```
In [79]: flights_df_no_cancelados = pd.DataFrame()
flights_df_no_cancelados = flights_df_no_cancelados_Temp_A.copy(deep=True)
```

```
In [80]: flights_df_no_cancelados_Temp_A = pd.DataFrame()
```

- Se verifican las aerolíneas

```
In [81]: flights_df_no_cancelados.head()
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPC
0	2015	1	1	4	AS	98	N407AS	A
1	2015	1	1	4	WN	1133	N627SW	C
2	2015	1	1	4	WN	1757	N752SW	I
3	2015	1	1	4	WN	1587	N8309C	I
4	2015	1	1	4	WN	4738	N8302F	I

```
In [82]: mask=(flights_df_no_cancelados['AIRLINE_IATA_CODE'].isnull()==True)  
flights_df_no_cancelados[mask]['YEAR'].count()
```

```
Out[82]: 0
```

```
In [83]: print("Cantidad de vuelos por Aerolínea:")  
print(flights_df_no_cancelados['AIRLINE_IATA_CODE'].value_counts())  
print()  
print("Cantidad total de vuelos:")  
print(flights_df_no_cancelados['AIRLINE_IATA_CODE'].count())
```

Cantidad de vuelos por Aerolínea:

```
WN    1242403  
DL    870275  
AA    712935  
OO    575386  
EV    554351  
UA    507716  
MQ    278791  
B6    262042  
US    194223  
AS    171315  
NK    115193  
F9    90090  
HA    75081  
VX    61248  
Name: AIRLINE_IATA_CODE, dtype: int64
```

Cantidad total de vuelos:

```
5711049
```

Se incorpora el nombre del aeropuerto

- [Volver](#)
- Se crea copia por si hay que volver atrás

```
In [84]: flights_df_no_cancelados_Temp_B = pd.merge(left = flights_df_no_cancelados, right = :
```

```
In [85]: flights_df_no_cancelados = pd.DataFrame()
flights_df_no_cancelados = flights_df_no_cancelados_Temp_B.copy(deep=True)
```

```
In [86]: flights_df_no_cancelados_Temp_B = pd.DataFrame()
```

- Se verifican los aeropuertos

```
In [87]: mask=(flights_df_no_cancelados['AIRPORT_IATA_CODE'].isnull()==True)
flights_df_no_cancelados[mask]['YEAR'].count()
```

```
Out[87]: 3871
```

- Quedaron 3871 registros con aeropuerto en nulo

```
In [88]: mask=flights_df_no_cancelados['AIRPORT_IATA_CODE'].isnull()==True
flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique()
```

```
Out[88]: array(['BSM'], dtype=object)
```

- Según https://es.wikipedia.org/wiki/Aeropuerto_Internacional_de_Austin-Bergstrom

Además de la comprobación anterior:

- Se comprueba que AUS no está para el mes de octubre y BSM si está presente
- Se comprueba que BSM no está presente para el resto de los meses pero si AUS
- Se comprueba que Operando ambos como origen tienen los mismos destinos
- Se comprueba que Operando ambos como destino tienen los mismos orígenes
- Se comprueba que AUS está presente en el dataset de aeropuertos, no así BSM
- Por lo tanto, se lo toma como un tema de reemplazo y no de eliminación, ya que "no" se generan registros duplicados y basando la desición en las comprobaciones realizadas habrá que reemplazar BSM por AUS

```
In [89]: mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="AUS") | (flights_df_no_cancelados['MONTH']==10))

print(f"AUS (=OCT): {flights_df_no_cancelados[mask].index.size}")

mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="BSM") | (flights_df_no_cancelados['MONTH']==10))

print(f"BSM (=OCT): {flights_df_no_cancelados[mask].index.size}")

AUS (=OCT): 0
BSM (=OCT): 7734
```

```
In [90]: mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="AUS") | (flights_df_no_cancelados['MONTH']!=10))

print(f"AUS (<>OCT): {flights_df_no_cancelados[mask].index.size}")

mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="BSM") | (flights_df_no_cancelados['MONTH']!=10))
```

```

print(f"BSM (<>OCT): {flights_df_no_cancelados[mask].index.size}")

AUS (<>OCT): 82837
BSM (<>OCT): 0

In [91]: mask=((flights_df_no_cancelados['ORIGIN_AIRPORT']=="AUS")) & \
          (flights_df_no_cancelados['MONTH']!=10)

print(f"AUS (ORIG):\n{sorted(flights_df_no_cancelados[mask]['DESTINATION_AIRPORT'])}

mask=((flights_df_no_cancelados['ORIGIN_AIRPORT']=="BSM")) & \
      (flights_df_no_cancelados['MONTH']==10)
print()
print(f"BSM (ORIG):\n{sorted(flights_df_no_cancelados[mask]['DESTINATION_AIRPORT'])}

AUS (ORIG):
['ATL', 'BNA', 'BOS', 'BWI', 'CLT', 'DAL', 'DCA', 'DEN', 'DFW', 'DTW', 'ELP', 'EW
R', 'FLL', 'HOU', 'HRL', 'IAD', 'IAH', 'JFK', 'LAS', 'LAX', 'LBB', 'LGB', 'MCO',
'MDW', 'MEM', 'MIA', 'MSP', 'MSY', 'OAK', 'ORD', 'PDX', 'PHL', 'PHX', 'SAN', 'SE
A', 'SFO', 'SJC', 'SLC', 'SNA', 'STL', 'TPA']

BSM (ORIG):
['ATL', 'BNA', 'BOS', 'BWI', 'CLT', 'DAL', 'DCA', 'DEN', 'DFW', 'DTW', 'ELP', 'EW
R', 'FLL', 'HOU', 'HRL', 'IAD', 'IAH', 'JFK', 'LAS', 'LAX', 'LBB', 'LGB', 'MCO',
'MDW', 'MIA', 'MSP', 'MSY', 'OAK', 'ORD', 'PHL', 'PHX', 'SAN', 'SEA', 'SFO', 'SJ
C', 'SLC', 'SNA', 'STL', 'TPA']

In [92]: mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="AUS")) & \
          (flights_df_no_cancelados['MONTH']!=10)

print(f"AUS (DEST):\n{sorted(flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique())}

mask=((flights_df_no_cancelados['DESTINATION_AIRPORT']=="BSM")) & \
      (flights_df_no_cancelados['MONTH']==10)
print()
print(f"BSM (DEST):\n{sorted(flights_df_no_cancelados[mask]['ORIGIN_AIRPORT'].unique())}

AUS (DEST):
['ATL', 'BNA', 'BOS', 'BWI', 'CLT', 'DAL', 'DCA', 'DEN', 'DFW', 'DTW', 'ELP', 'EW
R', 'FLL', 'HOU', 'HRL', 'IAD', 'IAH', 'JFK', 'LAS', 'LAX', 'LBB', 'LGB', 'MCO',
'MDW', 'MEM', 'MIA', 'MSP', 'MSY', 'OAK', 'ORD', 'PDX', 'PHL', 'PHX', 'SAN', 'SE
A', 'SFO', 'SJC', 'SLC', 'SNA', 'STL', 'TPA', 'TUL']

BSM (DEST):
['ATL', 'BNA', 'BOS', 'BWI', 'CLT', 'DAL', 'DCA', 'DEN', 'DFW', 'DTW', 'ELP', 'EW
R', 'FLL', 'HOU', 'HRL', 'IAD', 'IAH', 'JFK', 'LAS', 'LAX', 'LBB', 'LGB', 'MCO',
'MDW', 'MIA', 'MSP', 'MSY', 'OAK', 'ORD', 'PHL', 'PHX', 'SAN', 'SEA', 'SFO', 'SJ
C', 'SLC', 'SNA', 'STL', 'TPA']

In [93]: print(f"AUS:{airport_data[airport_data['IATA_CODE']=='AUS'].index.size}")
print(f"BSM:{airport_data[airport_data['IATA_CODE']=='BSM'].index.size}")

AUS:1
BSM:0

```

- Se crea copia por si hay que volver atrás

```
In [94]: flights_df_no_cancelados_Temp_Orig = flights_df_no_cancelados.copy(deep=True)
```

- Se borran columnas adicionadas

```
('AIRPORT_IATA_CODE','AIRPORT_DESCRIPTION','CITY','STATE','COUNTRY','LATITUDE','LONG
```

```
In [95]: flights_df_no_cancelados = flights_df_no_cancelados.drop(columns=['AIRPORT_IATA_CODE'])
```

- Se reemplaza valor BSM por AUS

```
In [96]: mask=(flights_df_no_cancelados['ORIGIN_AIRPORT']=='BSM')  
flights_df_no_cancelados[mask].to_csv(data_path + data_to_check + 'ORIGIN_AIRPORT_I  
flights_df_no_cancelados.loc[mask,'ORIGIN_AIRPORT']= 'AUS'  
  
flights_df_no_cancelados[mask]['YEAR'].index.size
```

```
Out[96]: 3871
```

```
In [97]: mask=(flights_df_no_cancelados['DESTINATION_AIRPORT']=='BSM')  
flights_df_no_cancelados[mask].to_csv(data_path + data_to_check + 'DESTINATION_AIRPORT_I  
#flights_df_no_cancelados.loc[mask,'DESTINATION_AIRPORT']= 'AUS'  
  
flights_df_no_cancelados[mask]['YEAR'].index.size
```

```
Out[97]: 3863
```

- Se vuelve a mergear aeropuerto

```
In [98]: flights_df_no_cancelados_Temp_C = pd.merge(left = flights_df_no_cancelados, right :
```

```
In [99]: flights_df_no_cancelados = pd.DataFrame()  
flights_df_no_cancelados = flights_df_no_cancelados_Temp_C.copy(deep=True)
```

```
In [100...]: flights_df_no_cancelados_Temp_C = pd.DataFrame()
```

```
In [101...]: flights_df_no_cancelados.head()
```

```
Out[101]:
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT
0	2015	1	1	4	AS	98	N407AS	A	D
1	2015	1	1	4	WN	1133	N627SW	E	I
2	2015	1	1	4	WN	1757	N752SW	I	
3	2015	1	1	4	WN	1587	N8309C	I	
4	2015	1	1	4	WN	4738	N8302F	I	

```
In [102...]: mask=(flights_df_no_cancelados['AIRPORT_IATA_CODE'].isnull()==True)  
flights_df_no_cancelados[mask]['YEAR'].count()
```

```
Out[102]: 0
```

```
In [103...]: flights_df_no_cancelados_Temp_Orig = pd.DataFrame()
```

Crear DF flights_retrasados

- [Volver](#)

```
In [104...]: mask = (flights_df_no_cancelados["ARRIVAL_DELAY"] > 0 )
flights_retrasados = flights_df_no_cancelados.loc[mask].copy(deep=True)
```

```
In [105...]: flights_retrasados.head()
```

```
Out[105]:
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPC
1	2015	1	1	4	WN	1133	N627SW	C
4	2015	1	1	4	WN	4738	N8302F	I
6	2015	1	1	4	WN	1908	N8324A	I
7	2015	1	1	4	WN	558	N522SW	M
8	2015	1	1	4	WN	257	N7742B	M

```
In [106...]: flights_retrasados.index.size
```

```
Out[106]: 2086045
```

Análisis

- [Volver](#)

Para que los textos sean interpretados de la misma manera por todos los lectores y poder homogeneizar conceptos, se tradujo los diferentes títulos y se buscó para los delays el significado de los mismos, en base a la siguiente url:

Fuente: <https://www.bts.gov/topics/airlines-and-airports/understanding-reporting-causes-flight-delays-and-cancellations>

Bureau of Transportation Statistics (Oficina de Estadísticas de Transporte)

- [Volver](#)

Columna Air System Delay

Air System Delay (NAS National Aviation System Delay): Delays and cancellations attributable to the national aviation system that refer to a broad set of conditions, such as non-extreme weather conditions, airport operations, heavy traffic volume, and air traffic control

Retraso del sistema de aire (Retraso del sistema de aviación nacional NAS): demoras y cancelaciones atribuibles al sistema nacional de aviación que se refieren a un amplio conjunto de condiciones, como condiciones climáticas no extremas, operaciones aeroportuarias, volumen de tráfico pesado y control de tráfico aéreo.

Columna Security Delay

Security Delay: Delays or cancellations caused by evacuation of a terminal or concourse, re-boarding of aircraft because of security breach, inoperative screening equipment and/or long lines in excess of 29 minutes at screening areas.

Retraso de seguridad: Demoras o cancelaciones causadas por la evacuación de una terminal o vestíbulo, reembarque de la aeronave debido a una violación de la seguridad, equipo de control inoperativo y/o largas filas de más de 29 minutos en las áreas de control.

Columna Airline Delay

Airline Delay (Air Carrier Delay): The cause of the cancellation or delay was due to circumstances within the airline's control (e.g. maintenance or crew problems, aircraft cleaning, baggage loading, fueling, etc.).

Retraso de la aerolínea (Retraso del transportista aéreo): La causa de la cancelación o demora se debió a circunstancias dentro del control de la aerolínea (p. ej., problemas de mantenimiento o de la tripulación, limpieza de la aeronave, carga de equipaje, abastecimiento de combustible, etc.).

Columna Late Aircraft Delay

Late Aircraft Delay (Aircraft Arriving Late): A previous flight with same aircraft arrived late, causing the present flight to depart late.

Retraso de aeronave tardía (Aeronaves que llegan tarde): un vuelo anterior con la misma aeronave llegó tarde, lo que provocó que el vuelo actual saliera tarde.

Columna Weather Delay

Weather Delay (Extreme Weather): Significant meteorological conditions (actual or forecasted) that, in the judgment of the carrier, delays or prevents the operation of a flight such as tornado, blizzard or hurricane.

Retraso del clima (Clima extremo): Condiciones meteorológicas significativas (reales o pronosticadas) que, a juicio del transportista, retrasan o impiden la operación de un vuelo, como tornados, ventiscas o huracanes.

También se buscó conceptos de aviación para entender el significado de algunas otras columnas.

Fuente: buscador Google

- **Taxi Out:** What does taxi out mean in aviation? The Taxi-out time is defined as the time spent by a flight between its actual off-block time (AOBT) and actual take-off time (ATOT). The Unimpeded taxi-out time is an average taxi-out time when there is no congestion. There is one unimpeded time by departure airport, departure runway and departure stand

¿Qué significa taxi out en aviación? El tiempo de rodaje de salida se define como el tiempo transcurrido por un vuelo entre su hora real fuera de calzos (AOBT) y la hora real de despegue (ATOT). El tiempo de rodaje de salida sin obstáculos es un tiempo medio de rodaje de salida cuando no hay congestión. Hay un tiempo sin obstáculos por el aeropuerto de salida, la pista de salida y el puesto de estacionamiento de salida.

- **Wheels Out:** In commercial and military aviation, wheels up describes an airborne plane or helicopter that has retracted its landing gear.

En la aviación comercial y militar, ruedas hacia arriba describe un avión o helicóptero en el aire que ha retraído su tren de aterrizaje.

Análisis de columnas de tiempo

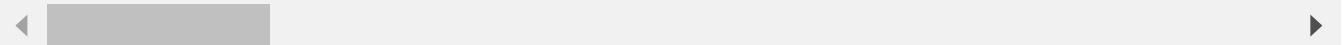
- [Volver](#)

Para comprender la significancia de los datos y la relación entre los mismos, se analizaron varios registros, de los cuales exponemos solo un caso a fin de simplificar.

```
In [107]: mask=(flights_df_no_cancelados['FLIGHT_NUMBER']==4141) & (flights_df_no_cancelados.flights_df_no_cancelados[mask]
```

```
Out[107]:
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT
69395	2015	1	5	1	EV	4141	N17196	RDU



* ORIGIN_AIRPORT = RDU	
* Scheluded departure (*Salida programada*)	8:15 HH:MM
* Departure Delay (*Retraso de salida*)	43 minutos
* Departure Time (*Hora de salida*)	8:58 HH:MM
* Taxi Out (*Salida de manga y carretero de subida*)	12 minutos
* Wheels Off (*Despegue, subida de ruedas*)	9:10 HH:MM
* Scheduled Time (*Tiempo estimado de vuelo*)	194 minutos
* Elapsed Time (*Tiempo transcurrido en Aire*)	178 minutos
(158+12+8)	
* Air Time (*tiempo aire*)	158 minutos
* Wheels On (Aterrizaje, Bajada de Ruedas)	10:48 HH:MM
(9:10 + 158(2:38)) (Nota)	
* Taxi In (Carretero de bajada y posición en manga)	8 minutos
* Arrival Time (Hora de llegada)	10:56 HH:MM
* Scheduled Arrival (Llegada Programada)	10:29 HH:MM

* Arrival Delay (Retraso de llegada)
* DESTINATION_AIRPORT = IAH

27 minutos

Nota: como se puede observar la suma de 9:10hs + 2:38hs da como resultado 11:48hs, no coincide con lo marcado por ***Wheels on***, esto es debido al cambio horario entre el origen y destino.

Se comprueba horarios de origen y destino

- [Volver](#)

In [108...]

```
print(IATA_TZ_data[IATA_TZ_data['iata_code']=='RDU']['time_zone'].item())
print(IATA_TZ_data[IATA_TZ_data['iata_code']=='IAH']['time_zone'].item())
```

America/New_York
America/Chicago

In [109...]

```
#RDU=America/New_York
#IAH=America/Chicago

start = datetime.datetime(2015, 1, 5, 9, 10)
BORDER_OUT=start + timedelta(minutes=int(158))
print(BORDER_OUT)

rng = pd.date_range(BORDER_OUT, periods=1, freq='D') #D Day Calendar daily
ts = pd.Series(np.random.randn(len(rng)), index=rng)

ts1 = ts.tz_localize('America/New_York')
ts2 = ts1[0:1].tz_convert('America/Chicago')

print(ts1.index)
print(ts2.index)
```

2015-01-05 11:48:00
DatetimeIndex(['2015-01-05 11:48:00-05:00'], dtype='datetime64[ns, America/New_York]', freq='D')
DatetimeIndex(['2015-01-05 10:48:00-06:00'], dtype='datetime64[ns, America/Chicago]', freq='D')

Se comprueba la unidad de distancia

- [Volver](#)

- RDU=America/New_York
- IAH=America/Chicago

Dato: Distance (*Distancia*) 1042

Se buscó en una página de viajes la distancia entre aeropuertos.

Fuente: <https://es.trip.com/flights/airport-iah-rdu/george-bush-intercontinental-airport-to-raleigh-international-airport/>

Arrojó una distancia 1.676,676 km

Se utilizó un conversor online de kilómetros a millas y devuelve: 1041.8381651

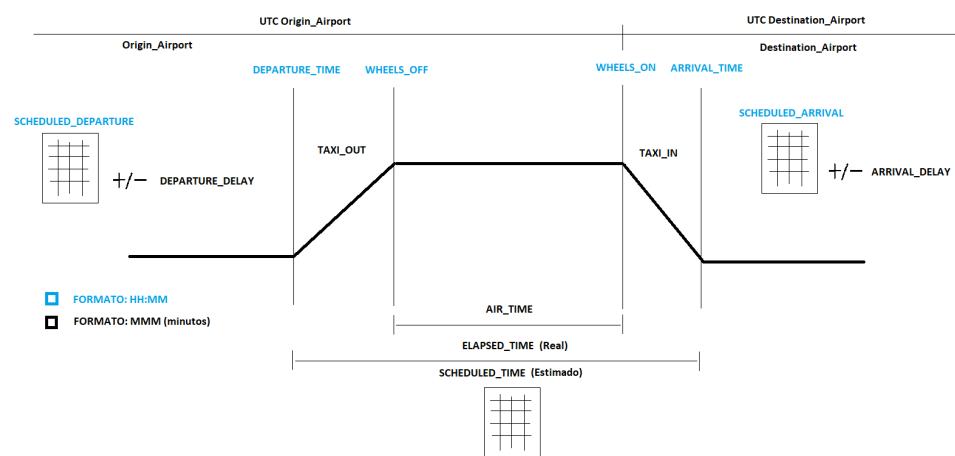
Fuente: [https://www.google.com/search? q=millas+a+km+distancia&rlz=1C1ASUM_enAR827AR828&oq=millas+a+km&aqs=chrome.1.68](https://www.google.com/search?q=millas+a+km+distancia&rlz=1C1ASUM_enAR827AR828&oq=millas+a+km&aqs=chrome.1.68)

Por lo tanto se concluye que las unidades vienen en millas.

Gráfica de Tiempos

- [Volver](#)

Gráfica de lo analizado con los datos hasta el momento:



Se verifica País

- [Volver](#)

- Verificar los países en el dataframe de Aeropuertos

```
In [110...]: countries = airport_data['COUNTRY'].value_counts().index
print("Total de países en archivo Aeropuertos: {}".format(len(countries)))
```

Total de países en archivo Aeropuertos: 1

- Verificar que país es, para chequear que se trate de USA

```
In [111...]: airport_data['COUNTRY'].value_counts().index
Out[111]: Index(['USA'], dtype='object')
```

Análisis de posible clave primaria

- [Volver](#)

- Contar el número de vuelos

```
In [112...]: flights_df_no_cancelados['FLIGHT_NUMBER'].count()
```

```
Out[112]: 5711049
```

- Contar por número de vuelo para ver si los mismos se repiten

```
In [113...]: flights_df_no_cancelados.groupby(['FLIGHT_NUMBER']).size()
```

```
Out[113]: FLIGHT_NUMBER
```

```
1      2360  
2      1961  
3      2848  
4      1754  
5      2242  
...  
8409     1  
8410     1  
8442     1  
8445     1  
9320     1  
Length: 6946, dtype: int64
```

Se verifica que sí, así que se toma un número de vuelo al azar para analizar. En este caso se toma el número de vuelo 1.

```
In [114...]: flights_df_no_cancelados[flights_df_no_cancelados['FLIGHT_NUMBER']==1]
```

Out[114]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
6787	2015	1	1		4	AA	1	N787AA
7448	2015	1	1		4	HA	1	N396HA
7688	2015	1	1		4	B6	1	N715JB
9321	2015	1	1		4	VX	1	N837VA
9573	2015	1	1		4	AS	1	N523AS
...
5704767	2015	12	31		4	AS	1	N588AS
5705433	2015	12	31		4	HA	1	N378HA
5705819	2015	12	31		4	VX	1	N843VA
5709560	2015	12	31		4	B6	1	N834JB
5709931	2015	12	31		4	AA	1	N785AA

2360 rows × 38 columns

Se vuelve a contar pero ahora por YEAR, MONTH, DAY, AIRLINE, FLIGHT_NUMBER, TAIL_NUMBER

Para ver si algún avión de la misma aerolínea realizó dos vuelos a la misma ciudad el mismo día

In [115...]

```
flights_df_no_cancelados.groupby(['YEAR', 'MONTH', 'DAY', 'AIRLINE', 'FLIGHT_NUMBEI
```

```
Out[115]:   YEAR  MONTH  DAY  AIRLINE  FLIGHT_NUMBER  TAIL_NUMBER
  2015      1       1    AA          1        N787AA         1
                  2        N795AA         1
                  3        N798AA         1
                  4        N799AA         1
                  5        N376AA         1
                         ..
  12       31    WN          6218        N925WN         1
                  6244        N8603F         2
                  6366        N387SW         3
                  6412        N967WN         1
                  6899        N386SW         3
```

Length: 4423617, dtype: int64

Sigue habiendo duplicados porque el total es: 5.711.049 contra 4.423.617 de la consulta anterior.

Se analiza más en profundidad uno de los casos en donde el origen da más de 1:

- 2015 12 31 WN 6366 N387SW

```
In [116...]: mask = (flights_df_no_cancelados['YEAR'] == 2015) & (flights_df_no_cancelados['MONTH'] == 12) & (flights_df_no_cancelados['DAY'] == 31) & (flights_df_no_cancelados['AIRLINE'] == 'WN') & (flights_df_no_cancelados['FLIGHT_NUMBER'] == 6366) & (flights_df_no_cancelados['TAIL_NUMBER'] == 'N387SW')
flights_df_no_cancelados[mask]
```

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
5698828	2015	12	31		4	WN	6366	N387SW
5705517	2015	12	31		4	WN	6366	N387SW
5708736	2015	12	31		4	WN	6366	N387SW

Se considera el ORIGIN_AIRPORT como parte de la clave. Se vuelve a contar pero ahora por:

- YEAR, MONTH, DAY, AIRLINE, FLIGHT_NUMBER, TAIL_NUMBER, ORIGIN_AIRPORT

```
In [117...]: flights_df_no_cancelados.groupby(['YEAR', 'MONTH', 'DAY', 'AIRLINE', 'FLIGHT_NUMBER', 'TAIL_NUMBER'])
```

	YEAR	MONTH	DAY	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	
2015	1	1	AA	1	1	N787AA	JFK	1
					2	N795AA	LAX	1
					3	N798AA	JFK	1
					4	N799AA	LAX	1
					5	N376AA	DFW	1
							..	
12	31	WN		6366	6366	N387SW	SJC	1
					6412	N967WN	EWR	1
					6899	N386SW	DAL	1
							HOU	1
							SAT	1

Length: 5711049, dtype: int64

Ahora se ve que se obtuvo una posible clave primaria para este dataset ya que arrojó la misma cantidad de registros que el analizado en un principio.

Esto no quita de tomar columnas adicionales si el origen también se duplicara. Pero para este conjunto con esta clave es suficiente.

Se verifica tiempo de vuelo máximo en todo el dataset

- [Volver](#)

```
In [118...]: print(flights_df_no_cancelados['ELAPSED_TIME'].max())
```

```
766
```

Se verifica condición de ARRIVAL_DELAY

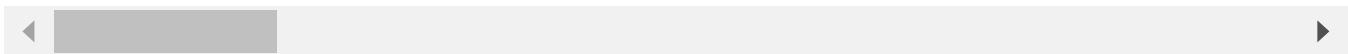
- [Volver](#)
- Se verifica cuales son los vuelos en que se cumple o no la siguiente condición:
ARRIVAL_DELAY = AIR_SYSTEM_DELAY + SECURITY_DELAY + AIRLINE_DELAY +
LATE_AIRCRAFT_DELAY + WEATHER_DELAY

```
In [119...]: mask = (flights_df_no_cancelados["ARRIVAL_DELAY"] > 0) & (flights_df_no_cancelados[
```

Out[119]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
4941724	2015	11	11	3	OO	5351	N956SW	
3350343	2015	8	2	7	UA	272	N455UA	
4910934	2015	11	9	1	WN	3143	N429WN	
830863	2015	2	26	4	DL	2011	N336NB	
300655	2015	1	21	3	US	654	N568UW	
...
5066922	2015	11	19	4	WN	2413	N559WN	
853711	2015	2	28	6	WN	4314	N704SW	
3286714	2015	7	29	3	WN	1708	N255WN	
3286725	2015	7	29	3	AA	861	N153UW	
1528792	2015	4	11	6	WN	1839	N8600F	

1022858 rows × 38 columns



In [120...]
`mask = (flights_df_no_cancelados["ARRIVAL_DELAY"] > 0) & (flights_df_no_cancelados.flights_df_no_cancelados.loc[mask].sort_values(by=['ARRIVAL_DELAY'], ascending=False))`

Out[120]:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN
340638	2015	1	23		5	AA	1322	N598AA
3343372	2015	8	1		6	AA	96	N479AA
4028202	2015	9	13		7	AA	1063	N3CAAA
5190016	2015	11	27		5	AA	2559	N489AA
5710442	2015	12	31		4	AA	2214	N4XKAA
...
115342	2015	1	8		4	WN	429	N938WN
3582524	2015	8	16		7	B6	477	N592JB
2985107	2015	7	11		6	WN	4586	N655WN
867881	2015	3	1		7	B6	2019	N296JB
2805233	2015	6	30		2	AA	2306	N3GGAA

1063187 rows × 38 columns

Analizar umbral a partir del cual computar una causa de retardo

- [Volver](#)
- Cuál es el umbral a partir del cual se comienza a computar una causa de retardo.
- Se comprueba si existe un posible umbral de informe

In [121...]

```
mask = (flights_df_no_cancelados['ARRIVAL_DELAY'] > 0) & ((flights_df_no_cancelados['AIR_SYSTEM_DELAY'] != 0) | (flights_df_no_cancelados['TAXI_DELAY'] != 0))
flights_df_no_cancelados[mask].sort_values("ARRIVAL_DELAY")['ARRIVAL_DELAY'].head(15)
```

Out[121]:

- Se comprueba si en el conjunto total pasa lo mismo

In [122...]

```
mask = (flights_data['ARRIVAL_DELAY'] > 0) & ((flights_data['AIR_SYSTEM_DELAY'] != 0) | (flights_data['TAXI_DELAY'] != 0))
flights_data[mask].sort_values("ARRIVAL_DELAY")['ARRIVAL_DELAY'].head(1).item()
```

Out[122]:

15.0

Dentro del conjunto de datos que almacena el dataset, existen registros que podrían clasificarse de la siguiente forma:

- Vuelos cancelados: CANCELLED = 1
- Vuelos derivados: DIVERTED = 1
- Vuelos llegados en horario: CANCELLED = 0, DIVERTED = 0, ARRIVAL_DELAY = 0.
- Vuelos llegados previo al horario planificado: CANCELLED = 0, DIVERTED = 0, ARRIVAL_DELAY < 0.
- Vuelos llegados posterior al horario planificado: CANCELLED = 0, DIVERTED = 0, ARRIVAL_DELAY > 0.

Ahora bien, yendo puntualmente a los vuelos con llegada posterior al horario planificado, según lo visto en los registros de vuelos, existe un umbral el cual determina a partir de cuantos minutos de retrasos resulta necesario tener que “explicar” el mismo.

Analizando el dataset se comprobó que este umbral es de 15 minutos. Menor a 15 minutos, solo se registra el delay en ARRIVAL_DELAY, sin tener la necesidad de asignar estos minutos a ninguna de las 5 causas (AIR_SYSTEM_DELAY, SECURITY_DELAY, AIRLINE_DELAY, LATE_AIRCRAFT_DELAY, WEATHER_DELAY). En cambio, a partir de los 15 minutos, cada registro divide en una o más causas los minutos de retraso, debiendo esta división llegar al total registrado en la columna ARRIVAL_DELAY.

Dicho esto, se tomó la decisión de focalizar el análisis en vuelos con delay (ARRIVAL_DELAY > 0), que no estén cancelados (CANCELLED = 0) ni derivados (DIVERTED = 0).

Visualizaciones

- [Volver](#)

¿Qué datos deberían tenerse en cuenta antes de contratar un vuelo?

- [Volver](#)

Como consumidores de vuelos no hay nada que afecte más nuestro humor que tener que esperar en un aeropuerto.

Partiendo de la base de que muchas veces las personas no pueden planificar o elegir el momento exacto en que quisiera tomar un vuelo dado que son muchos los factores al momento de tomar la decisión, el análisis de centrará en determinar en qué época del año debiéramos viajar sin tener que esperar mucho tiempo por inconvenientes por delay.

Es por esto que, en función a la información que nos brinda el dataset, mostraremos datos que cualquier usuario final debería revisar previo a contratar un ticket aéreo.

Aclaración previa: los datos trabajados, tal como se mencionó, corresponden al año 2015 por lo que se hace la aclaración que, las conclusiones que se formularán, las hacemos considerando como si año tras año, las mismas se repetieran.

In [123...]

```
def truncate_format(number: float, max_decimals: int) -> (str, float):
    """
    Desc: Función que dado el número flotante y un cantidad de decimales
          Trunca el número a la cantidad de decimales, pero no redondea.
          También devuelve su representación en string.

    Ej: number=23.4567 max_decimals=2
        out "23.45" 23.45

        number=1.2 max_decimals=3
        out "1.200" 1.2

    Args:
        number (float): número a formatear
        max_decimals (int): máxima cantidad de decimales

    Returns:
        (a,b) (str, float):
            a= string con la representación del número truncado
            b= número flotante
    """

    dec=""
    for n in range(max_decimals):
        dec+="0"

    if type(number) == type(3):
        if max_decimals <= 0:
            a = str(number)
        else:
            a = ".".join((str(number), dec))
    else:
        int_part, dec_part = str(number).split(".")
        cadena = dec_part + dec
        if max_decimals <= 0:
            a = int_part
        else:
            a = ".".join((int_part, cadena[:max_decimals]))

    b = np.float64(a)

    return a,b
```

In [124...]

```
def show_values(axs=None, orient="v", graficas=None, view_float=True, count_dec=0,
               **kwargs):
    """
    Desc: Función que muestra los valores de las barras en el gráfico de barras.
          Las barras pueden ser horizontales o verticales, solapadas (no apiladas)

    Barras verticales: en dirección al eje x, la altura del texto de los valores
    calcula automáticamente desde el plot más cercano al eje x hasta el plot
    (no importa el cambio de límite en el eje "y" o de dpi en la figura)
    En dirección opuesta si el límite superior es más chico que el máximo valor
    avisa con mensaje para reajustarla en forma manual.

    Si el valor no entra desde el edge superior de la barra, lo reajusta a una
    a partir de la horizontal del eje x.

    Consideraciones: Se asume un proporción de letra de 0.85 entre su alto y
```

Barras horizontales: misma funcionalidad y consideraciones.

Args:

```
    axs (AxesSubplot): AxesSubplot a modificar
    orient (str(v o h)): Como calculará la posición de los valores
        (Tener en cuenta si el gráfico de barras es horizontal)
    graficas (int): cantidad de plots en el AxesSubplot
    view_float (bool): Ver los valores con formato entero o flotante
    count_dec (int): si view_float es true, cuantos decimales llevarán los valores
    text_rotation (int): Rotación del texto Ej: 90
    intervalo (tupla de enteros): índice (desde-hasta) de valores a visualizar
```

Returns:

```
    Aplica los cambios en el mismo AxesSubplot pasado como argumento
```

```
"""
try:
```

```
    tmp = axs.transData.transform([(0,0), (1,1)])
    if orient == "v":
        tmp = tmp[1,1] - tmp[0,1] # 1 unit in display coords
    else:
        tmp = tmp[1,0] - tmp[0,0] # 1 unit in display coords
    tmp2 = 1/tmp # 1 pixel in display coords
    dy=0 # off-set
    dpi_100=axs.get_figure().get_dpi()/100 # Proporción del dpi actual respecto al 100
    tmp = tmp2*dy*axs.get_figure().get_dpi() # shift pixels in display coords
    p_letra=0.85 # proporción de las letras entre su ancho y alto (A mejorar)
```

```
    cant=len(axs.patches) #Cantidad de rectángulos
    cantXgraf=int(cant/graficas) #Cantidad de rectangulos por plot
    rectangles=np.arange(cant) #Array de índices
    texts = rectangles.reshape(graficas, cantXgraf) #Array de índices por plot
```

```
    if orient == "v":
```

```
        #-----
        # Generar todos los texts con su reajuste de edge
        #-----
```

```
        una_vez=True
```

```
        for i in rectangles:
            p=axs.patches[i]
```

```
            _x=p.get_x() + p.get_width()/2
            _y0=p.get_y() + p.get_height() + tmp
            _s=str(p.get_height())
```

```
            if (legends is None):
                if (view_float==True):
                    _s, _v = truncate_format(p.get_height(), count_dec)
                else:
                    _s, _v = truncate_format(p.get_height(), 0)
            else:
                _s=legends[i]
```

```
            texto=axs.text(x=_x, y=_y0, s=_s, rotation=text_rotation, va='top'
            _y1=(p.get_y() + p.get_height())-((texto.get_size()*p_letra)*tmp2)
```

```
            lim_sup=axs.get_ylim()[1]
```

```
# Reajustes
```

```
if lim_sup<_y0:
    if una_vez:
        una_vez=False
        print("Límite superior erróneo, aumente el valor del límite")
    _y0 = lim_sup
```

```

        texto.set_y(_y0)

    if _y1<=0:
        _y0 = _y0 - _y1
        texto.set_y(_y0)

#-----
# Reposicionar para que no haya superposición
#-----
for n in list(range(graficas-2,-1,-1)):
    indices=texts[n]
    for i in indices:
        p=axs.patches[i]

        _y1=(p.get_y() + p.get_height())-((axs.texts[i].get_size()*p_letra)*tmp2)
        dif=_y1-axs.texts[i+cantXgraf].get_position()[1]
        if dif<0:
            axs.texts[i].set_y(axs.texts[i].get_position()[1]-dif)

    elif orient == "h":

#-----
# Generar todos los texts con su reajuste de edge
#-----
una_vez=True
for i in rectangles:
    p=axs.patches[i]

    _y=p.get_y() + p.get_height()/2
    _x0=p.get_x() + p.get_width() + tmp
    _s=str(p.get_width())

    if (legends is None):
        if (view_float==True):
            _s, _v = truncate_format(p.get_width(), count_dec)
        else:
            _s, _v = truncate_format(p.get_width(), 0)
    else:
        _s=legends[i]

    texto=axs.text(x=_x0, y=_y, s=_s, rotation=text_rotation, va='center')
    _x1=(p.get_x() + p.get_width())-((texto.get_size()*p_letra)*tmp2*_limite)
    lim_sup=axs.get_xlim()[1]

    # Reajustes
    if lim_sup<_x0:
        if una_vez:
            una_vez=False
            print(f'Límite superior erróneo, aumente el valor del límite')
        _x0 = lim_sup
        texto.set_x(_x0)

    if _x1<=0:
        _x0 = _x0 - _x1
        texto.set_x(_x0)

#-----
# Reposicionar para que no haya superposición
#-----
for n in list(range(graficas-2,-1,-1)):
    indices=texts[n]
    for i in indices:

```

```

        p=axs.patches[i]

        _x1=(p.get_x() + p.get_width())-((axs.texts[i].get_size()*p_le-
        dif=_x1-axs.texts[i+cantXgraf].get_position()[1]
        if dif<0:
            axs.texts[i].set_x(axs.texts[i].get_position()[1]-dif)

#-----
# Visualizar textos
#-----
for i in rectangles:
    if (i >= intervalo[0]) and (i <= intervalo[1]):
        axs.texts[i].set_visible(True)

except Exception as e:
    print(f"Error al tratar de mostrar los valores de las gráficas")
    print(e)
    print("")

```

Análisis Mensual

- [Volver](#)

In [125...]

```

def f_mes_agrup_porc(x, airline='All'):
    '''
    Esta función filtra por aerolínea y agrega las columnas de PORC_RETRASOS y PORC_VUELOS.

    Parameters
    -----
    x : dataframe
        datos mensuales de las aerolíneas.

    airline: str
        "All" o un AIRLINE_DESCRIPTION

    Returns
    -----
    resultados: dataframe
        con las columnas de porcentajes de vuelos y retrasos y los datos filtrados

    Raises
    -----
    Exception
        Si 'airline' es distinto de 'All', o algún valor de AIRLINE_DESCRIPTION.

    Notes
    -----
    ...

try:

    if airline=='All':
        data_temp = x.copy(deep=True)
        data_temp.drop(columns=['AIRLINE_DESCRIPTION'])
        data_temp = data_temp.groupby(['MONTH'])[['VUELOS', 'RETRASOS']].sum()
    else:
        mask=(x['AIRLINE_DESCRIPTION']==airline)
        data_temp = x[mask].copy(deep=True)
        data_temp.set_index('MONTH', drop=True, inplace=True)

```

```

        data_temp['PORC_RETRASOS'] = round((data_temp.RETRASOS / data_temp.VUELOS) * 100, 2)
        data_temp['PORC_VUELOS'] = round((data_temp.VUELOS / data_temp.VUELOS) * 100, 2)

    return data_temp

except Exception as e:
    print(f"Error al tratar de preparar los datos mensuales. airline:{airline}")
    print(e)
    print("")

```

```

In [126...]: def f_mes_prep_vis(airline):
    flights_month_delay = pd.DataFrame({
        'VUELOS' : flights_df_no_cancelados.groupby(['MONTH', 'AIRLINE_DESCRIPTION']).sum(),
        'RETRASOS' : flights_retrasados.groupby(['MONTH', 'AIRLINE_DESCRIPTION']).sum()
    }).reset_index()

    flights_month_delay['RETRASOS'].replace(np.nan, 0, inplace = True)

    flights_month_delay_porc = f_mes_agrup_porc(flights_month_delay, airline=airline)

    if flights_month_delay_porc.empty:
        print("No existen datos")
    else:
        f_mes_vis(airline, flights_month_delay_porc)

```

```

In [127...]: def f_mes_vis(airline, data):
    sns.set_color_codes("pastel")
    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(8,6), layout='constrained',
                           sharex=True)

    top_bar = mpatches.Patch(color='b', label='Total vuelos')
    bottom_bar = mpatches.Patch(color='r', label='Con retraso')

    graficas=2
    count_view= len(data.index)
    sns.barplot(x=data.index,y="PORC_VUELOS", data=data, color="b", ax=axs[0])
    sns_barplot(x=data.index,y="PORC_RETRASOS", data=data, color="r", ax=axs[0])
    axs[0].set_title(f'% Retrasos por mes a lo largo del año\n'{airline}'), size=12
    axs[0].set_xticks([0,1,2,3,4,5,6,7,8,9,10,11])
    axs[0].set_xticklabels(['Ene','Feb','Mar','Abr','May','Jun','Jul','Ago','Sep','Oct','Nov','Dic'])
    axs[0].set_xlabel(''), ylabel='')

    show_values(axs=axs[0], graficas=graficas, view_float=True, count_dec=2, text_color='black')
    axs[0].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=[0.0, 1.0])

    sns_barplot(x=data.index,y="VUELOS", data=data, color="b", ax=axs[1])
    sns_barplot(x=data.index,y="RETRASOS", data=data, color="r", ax=axs[1])
    axs[1].set_title(f'Retrasos por mes a lo largo del año\n'{airline}), size=12
    axs[1].set_xticks([0,1,2,3,4,5,6,7,8,9,10,11])
    axs[1].set_xticklabels(['Ene','Feb','Mar','Abr','May','Jun','Jul','Ago','Sep','Oct','Nov','Dic'])
    axs[1].set_xlabel(''), ylabel='')

    show_values(axs=axs[1], graficas=graficas, view_float=False, text_rotation=90, count_dec=2)
    axs[1].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=[0.0, 1.0])

plt.show()

```

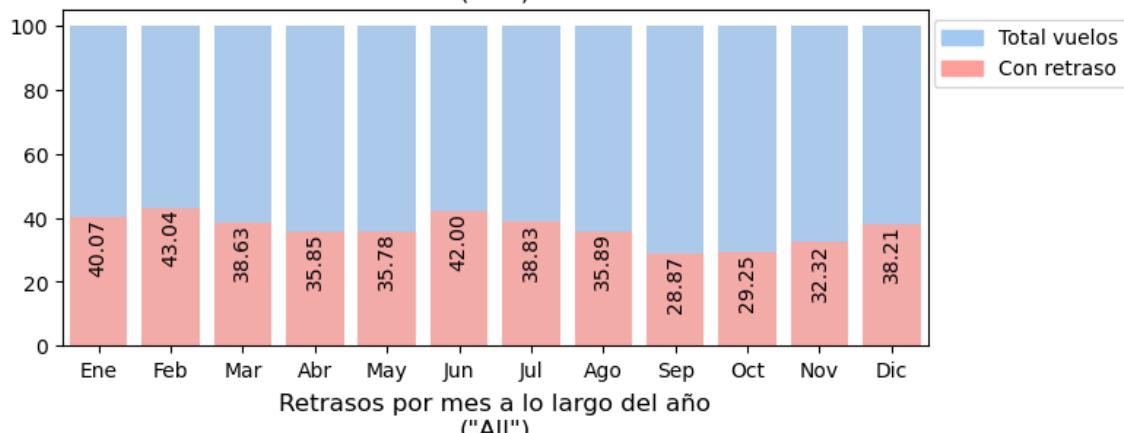
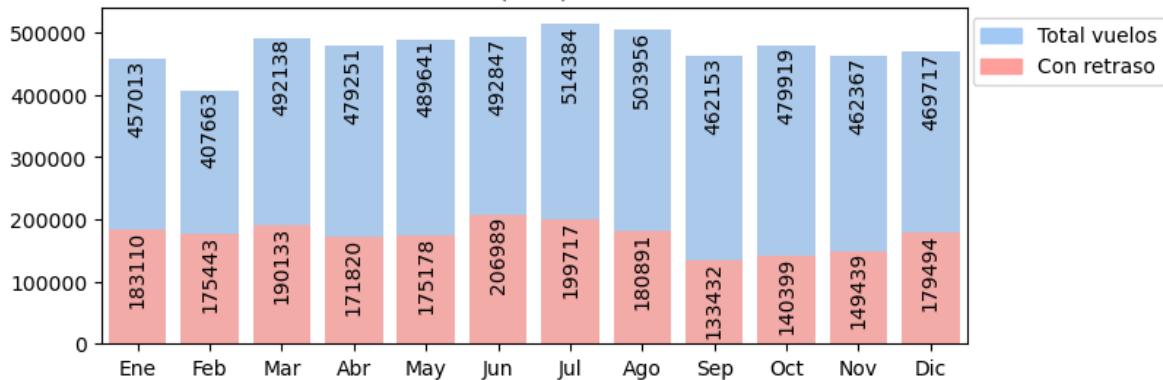
```

In [128...]: w = interactive( f_mes_prep_vis, airline=np.insert(flights_df_no_cancelados.AIRLINES, 0, 'Iberia'))
display(w)

```

airline

All

% Retrasos por mes a lo largo del año
("All")Retrasos por mes a lo largo del año
("All")

In [129...]

```
flights_month_delay = pd.DataFrame()
flights_month_delay_porc = pd.DataFrame()
```

Conclusión 'Mensual'

De los gráficos se puede ver que en Febrero se tiene la menor cantidad de vuelos, mientras que en Julio se tiene la más alta.

Cercano a este último, se encuentran los meses de Junio y Agosto en segundo y tercer puesto respectivamente. Es posible atribuir este aumento de oferta al periodo vacacional del país.

Ahora bien, si nos enfocamos en vuelos con delay, Junio es quien mayor número tiene.

Respecto a los % de ocurrencia, febrero es el mes que más tiene, seguido de Junio.

En lo que a tráfico aéreo respecta, no sigue la lógica comúnmente pensada de mayor tráfico, mayor delay.

Tener además en cuenta que si bien el % de ocurrencia de retrasos en febrero es mayor a nivel global, para algunas aerolíneas en particular el % puede ser menor con respecto a otros meses de esa misma aerolínea.

Lo mismo ocurre con septiembre, el % de ocurrencia de demora es menor a nivel global pero para algunas aerolíneas puede ser mayor respecto a otros meses.

Por tal motivo recomendamos consultar las demoras por aerolínea

Análisis diario (día del mes)

- [Volver](#)

In [130...]

```
def f_dia_mes_agrup_porc(x, year=2015, month='All', airline='All'):
    """
    Esta función filtra por aerolínea y agrega las columnas de PORC_RETRASOS y PORC_VUELOS.

    Parameters
    -----
    x : dataframe
        datos mensuales de las aerolíneas.

    airline: str
        "All" o un AIRLINE_DESCRIPTION

    Returns
    -----
    resultados: dataframe
        con las columnas de porcentajes de vuelos y retrasos y los datos filtrados

    Raises
    -----
    Exception
        Si 'airline' es distinto de 'All', o algún valor de AIRLINE_DESCRIPTION.

    Notes
    -----
    ...

try:

    #Filtrar por año al menos 1 tiene que haber
    mask=(x['YEAR']==np.int64(year))
    data_temp=x[mask].copy(deep=True)

    if airline=='All':
        data_temp.drop(columns=['AIRLINE_DESCRIPTION'], inplace=True)
        #month 0 = All
        if month==0:
            data_temp.drop(columns=['MONTH'], inplace=True)
        else:
            mask=(data_temp['MONTH']==month)
            data_temp = data_temp[mask].copy(deep=True)
            data_temp.drop(columns=['MONTH'], inplace=True)
    else:
        mask=(x['AIRLINE_DESCRIPTION']==airline)
        data_temp = data_temp[mask].copy(deep=True)
        data_temp.drop(columns=['AIRLINE_DESCRIPTION'], inplace=True)
        #month 0 = All
        if month==0:
            data_temp.drop(columns=['MONTH'], inplace=True)
        else:
            mask=(data_temp['MONTH']==month)
            data_temp = data_temp[mask].copy(deep=True)
            data_temp.drop(columns=['MONTH'], inplace=True)
    data_temp = data_temp.groupby(['DAY'])[['VUELOS', 'RETRASOS']].sum()

    data_temp['PORC_RETRASOS'] = round((data_temp.RETRASOS/data_temp.VUELOS)*100,2)
    data_temp['PORC_VUELOS'] = round((data_temp.VUELOS/data_temp.VUELOS)*100,2

    return data_temp

except Exception as e:
    print(f"Error al tratar de preparar los datos por día del mes. year:{year}")
    print(e)
    print("")
```

```
In [131...]  

def f_dia_mes_prep_vis(cb_year, cb_month, cb_airline):  

    flights_day_month_delay = pd.DataFrame({  

        'VUELOS' : flights_df_no_cancelados.groupby(['YEAR', 'MONTH', 'DAY', 'AIRLINE_ID'])['VUELOS'].sum(),  

        'RETRASOS' : flights_retrasados.groupby(['YEAR', 'MONTH', 'DAY', 'AIRLINE_ID'])['RETRASOS'].sum()  

    }).reset_index()  

    flights_day_month_delay['RETRASOS'].replace(np.nan, 0, inplace = True)  

    flights_day_month_delay_porc = f_dia_mes_agrup_porc(flights_day_month_delay, year=cb_year, month=cb_month, airline=cb_airline)  

    if flights_day_month_delay_porc.empty:  

        print("No existen datos")  

    else:  

        f_dia_mes_vis(cb_year, cb_month, cb_airline, flights_day_month_delay_porc)
```

```
In [132...]  

def f_dia_mes_vis(year, month, airline, data):  

    sns.set_color_codes("pastel")  

    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(8,6), layout='constrained',  

                           sharex=True)  

    top_bar = mpatches.Patch(color='b', label='Total vuelos')  

    bottom_bar = mpatches.Patch(color='r', label='Con retraso')  

    days=[]  

    for n in range(0,32,1):  

        days.append(n)  

    graficas=2  

    count_view= len(data.index)  

    sns.barplot(x=data.index,y="PORC_VUELOS", data=data, color="b", ax=axs[0])  

    sns_barplot = sns.barplot(x=data.index,y="PORC_RETRASOS", data=data, color="r", ax=axs[0])  

    axs[0].set_title(f'% Retrasos por día a lo largo del año\n{year}/{months_dic[month]}')  

    axs[0].set_xticks(days[:31])  

    axs[0].set_xticklabels(days[1:])  

    axs[0].set(xlabel='', ylabel='')  

    show_values(axs=axs[0], graficas=graficas, view_float=True, count_dec=2, text_color='black')  

    axs[0].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=(0, 1))  

    sns_barplot = sns.barplot(x=data.index,y="VUELOS", data=data, color="b", ax=axs[1])  

    sns_barplot = sns.barplot(x=data.index,y="RETRASOS", data=data, color="r", ax=axs[1])  

    axs[1].set_title(f'Retrasos por día a lo largo del año\n{year}/{months_dic[month]}')  

    axs[1].set_xticks(days[:31])  

    axs[1].set_xticklabels(days[1:])  

    axs[1].set(xlabel='', ylabel='')  

    show_values(axs=axs[1], graficas=graficas, view_float=False, text_rotation=90, text_color='white')  

    axs[1].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=(0, 1))  

    plt.show()
```

```
In [133...]  

cb_airline = widgets.Dropdown(options=np.insert(flights_df_no_cancelados.AIRLINE_ID.unique(), 0, 'Todos'), value='Todos', description='Aerolinea')  

cb_month = widgets.Dropdown(options=months, value=0, description='Mes')  

cb_year = widgets.Dropdown(options=['2015'], description='Año', disabled=True)  

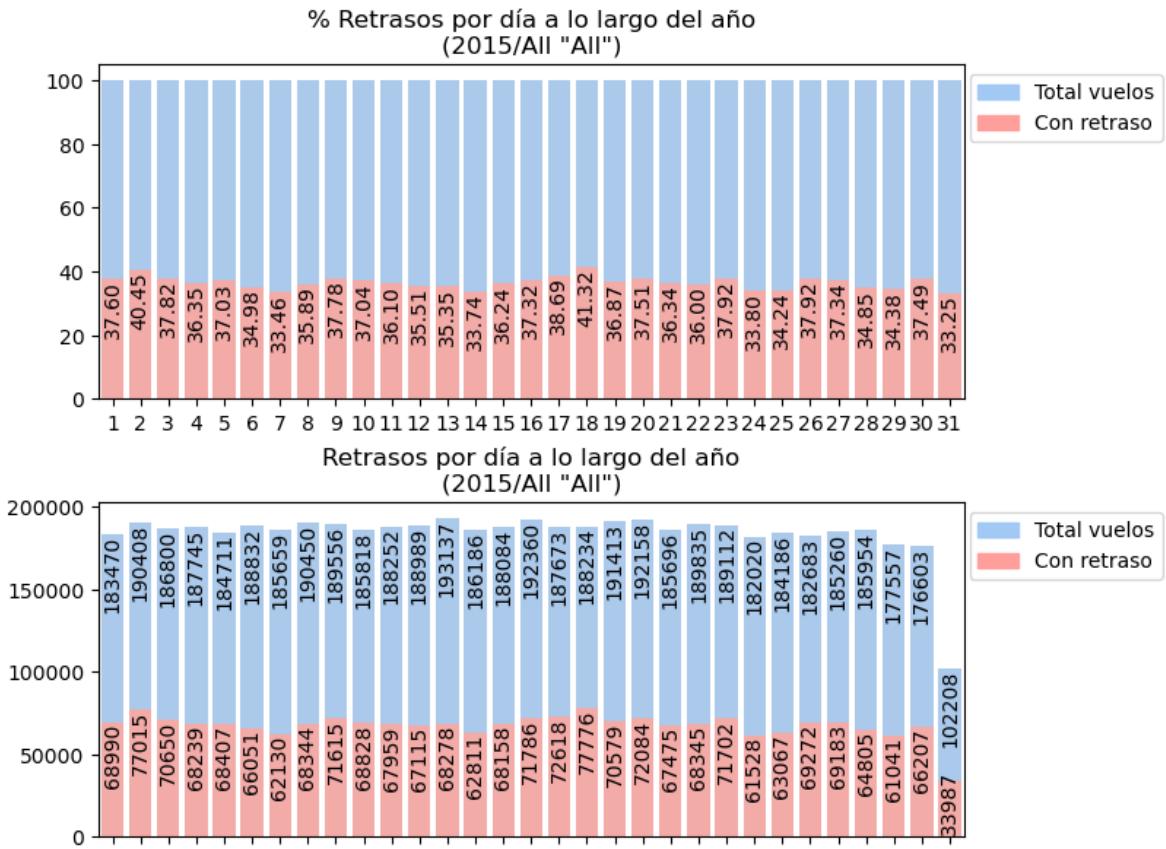
ui = widgets.HBox([cb_year, cb_month, cb_airline])  

out = widgets.interactive_output(f_dia_mes_prep_vis, {'cb_year':cb_year, 'cb_month':cb_month, 'cb_airline':cb_airline})  

display(ui, out)
```

Año Mes



```
In [134...]: flights_day_month_delay = pd.DataFrame()
flights_day_month_delay_porc = pd.DataFrame()
```

Conclusión 'diario (día del mes)'

Estos gráficos muestran un número estable de vuelos durante los días del mes. Nominalmente no existen marcadas diferencias en la oferta de vuelos. En lo que respecta al delay, la conclusión es similar. Los picos se dan al principio y mitad de mes. Es importante mencionar la salvedad que los números indicados para los últimos días del mes pueden ser afectados dado que no todos los meses tienen la misma cantidad de días.

Análisis diario (día de la semana)

- [Volver](#)

```
In [135...]: def f_dia_sem_agrup_porc(x, year=2015, month='All', airline='All'):
    ...
    Esta función filtra por aerolínea y agrega las columnas de PORC_RETRASOS y PORC
    Returns
    -----
    Parameters
    -----
    x : dataframe
        datos mensuales de las aerolíneas.

    airline: str
        "All" o un AIRLINE_DESCRIPTION

    Returns
    -----
    resultados: dataframe
```

```

    con las columnas de porcentajes de vuelos y retrasos y los datos filtrados

Raises
-----
Exception
    Si 'airline' es distinto de 'All', o algún valor de AIRLINE_DESCRIPTION.

Notes
-----
...
try:

#Filtrar por año al menos 1 tiene que haber
mask=(x['YEAR']==np.int64(year))
data_temp=x[mask].copy(deep=True)

if airline=='All':
    data_temp.drop(columns=['AIRLINE_DESCRIPTION'], inplace=True)
#month 0 = All
if month==0:
    data_temp.drop(columns=['MONTH'], inplace=True)
else:
    mask=(data_temp['MONTH']==month)
    data_temp = data_temp[mask].copy(deep=True)
    data_temp.drop(columns=['MONTH'], inplace=True)
else:
    mask=(x['AIRLINE_DESCRIPTION']==airline)
    data_temp = data_temp[mask].copy(deep=True)
    data_temp.drop(columns=['AIRLINE_DESCRIPTION'], inplace=True)
#month 0 = All
if month==0:
    data_temp.drop(columns=['MONTH'], inplace=True)
else:
    mask=(data_temp['MONTH']==month)
    data_temp = data_temp[mask].copy(deep=True)
    data_temp.drop(columns=['MONTH'], inplace=True)

data_temp = data_temp.groupby(['DAY_OF_WEEK'])[['VUELOS', 'RETRASOS']].sum

data_temp['PORC_RETRASOS'] = round((data_temp.RETRASOS/data_temp.VUELOS)*100,2)
data_temp['PORC_VUELOS'] = round((data_temp.VUELOS/data_temp.VUELOS)*100,2)

return data_temp

except Exception as e:
    print(f"Error al tratar de preparar los datos por día de la semana. airline={airline}")
    print(e)
    print("")

```

```

In [136]: def f_dia_sem_prep_vis(cb_year, cb_month, cb_airline):
    flights_day_week_delay = pd.DataFrame({
        'VUELOS' : flights_df_no_cancelados.groupby(['YEAR', 'MONTH', 'DAY_OF_WEEK']),
        'RETRASOS' : flights_retrasados.groupby(['YEAR', 'MONTH', 'DAY_OF_WEEK', 'DAY_OF_WEEK']))
    }).reset_index()

    flights_day_week_delay['RETRASOS'].replace(np.nan, 0, inplace = True)

    flights_day_week_delay_porc = f_dia_sem_agrup_porc(flights_day_week_delay, year=cb_year, month=cb_month, airline=cb_airline)

    if flights_day_week_delay_porc.empty:
        print("No existen datos")
    else:
        f_dia_sem_vis(cb_year, cb_month, cb_airline, flights_day_week_delay_porc)

```

```
In [137...]
def f_dia_sem_vis(year, month, airline, data):
    sns.set_color_codes("pastel")
    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(8,5), layout='constrained')

    top_bar = mpatches.Patch(color='b', label='Total vuelos')
    bottom_bar = mpatches.Patch(color='r', label='Con retraso')

    days=[]
    for n in range(0,7,1):
        days.append(n)

    graficas=2
    count_view= len(data.index)
    sns.barplot(x=data.index,y="PORC_VUELOS", data=data, color="b", ax=axs[0])
    sns.legend_0 = sns.barplot(x=data.index,y="PORC_RETRASOS", data=data, color="r", ax=axs[0])
    axs[0].set_title(f'% Retrasos por día de la semana a lo largo del año\n{year}',)
    axs[0].set_xticks(days)
    axs[0].set_xticklabels(['Lun', 'Mar', 'Mie', 'Jue', 'Vie', 'Sab','Dom'])
    axs[0].set_xlabel(''), ylabel='')
    show_values(axs=axs[0], graficas=graficas, view_float=True, count_dec=2, text_
    axs[0].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor : 

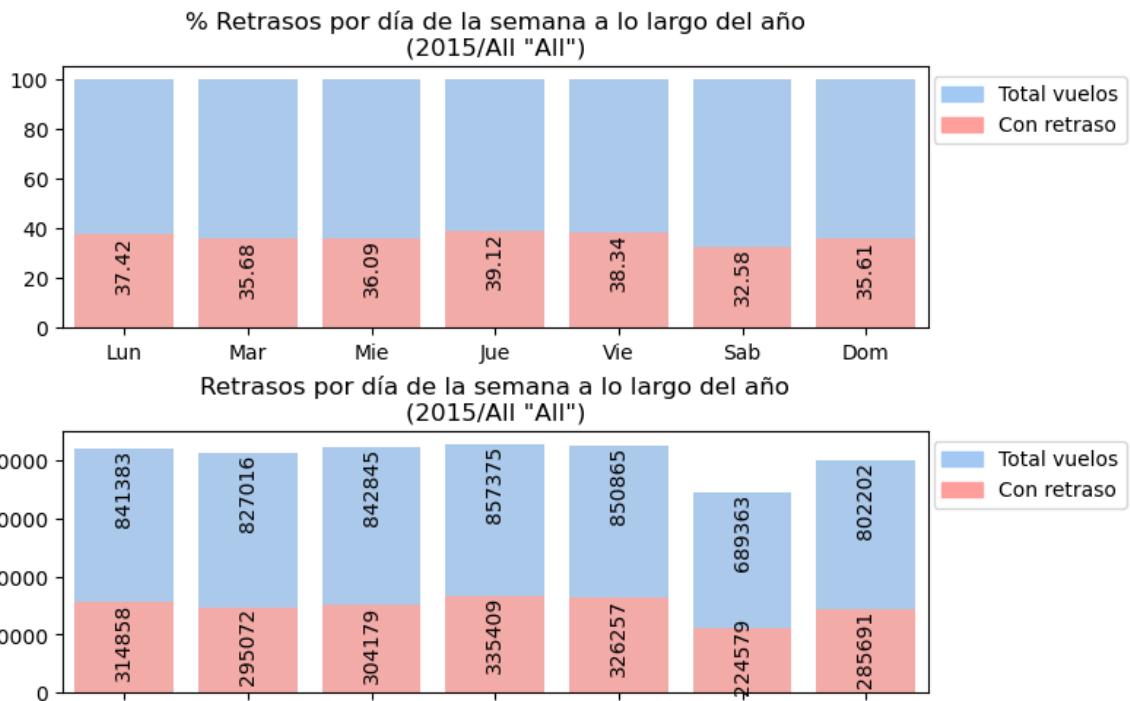
    sns.barplot(x=data.index,y="VUELOS", data=data, color="b", ax=axs[1])
    sns.barplot(x=data.index,y="RETRASOS", data=data, color="r", ax=axs[1])
    axs[1].set_title(f'Retrasos por día de la semana a lo largo del año\n{year}/')
    axs[1].set_xticks(days)
    axs[1].set_xticklabels(['Lun', 'Mar', 'Mie', 'Jue', 'Vie', 'Sab','Dom'])
    axs[1].set_xlabel(''), ylabel='')
    show_values(axs=axs[1], graficas=graficas, view_float=False, text_rotation=90,
    axs[1].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor : 

    plt.show()
```

```
In [138...]
cb_airline = widgets.Dropdown(options=np.insert(flights_df_no_cancelados.AIRLINE_DI
cb_month = widgets.Dropdown(options=months, value=0, description='Mes')
cb_year = widgets.Dropdown(options=['2015'], description='Año', disable=True)
ui2 = widgets.HBox([cb_year, cb_month, cb_airline])

out2 = widgets.interactive_output(f_dia_sem_prep_vis, {'cb_year':cb_year, 'cb_mont
display(ui2, out2)
```

Año Mes



```
In [139...]: flights_day_week_delay = pd.DataFrame()
flights_day_week_delay_porc = pd.DataFrame()
```

Conclusión (día de la semana)

Los gráficos muestran que el número de vuelos durante los días de semana y domingo se mantienen estables, mientras que los sábados hay una disminución.

Análisis horario

- [Volver](#)

```
In [140...]: def f_hora_agrup_porc(x, year=2015, airline='All'):
    ...
    Esta función filtra por aerolínea y agrega las columnas de PORC_RETRASOS y PORC_VUELOS.

    Parameters
    -----
    x : dataframe
        datos mensuales de las aerolíneas.

    airline: str
        "All" o un AIRLINE_DESCRIPTION

    Returns
    -----
    resultados: dataframe
        con las columnas de porcentajes de vuelos y retrasos y los datos filtrados

    Raises
    -----
    Exception
        Si 'airline' es distinto de 'All', o algún valor de AIRLINE_DESCRIPTION.

    Notes
    -----
```

```

...
try:

    #Filtrar por año al menos 1 tiene que haber
    mask=(x['YEAR']==np.int64(year))
    data_temp=x[mask].copy(deep=True)

    if airline=='All':
        data_temp.drop(columns=['AIRLINE_DESCRIPTION'], inplace=True)
    else:
        mask=(data_temp['AIRLINE_DESCRIPTION']==airline)
        data_temp = data_temp[mask].copy(deep=True)

    data_temp = data_temp.groupby(['HOUR'])[['VUELOS', 'RETRASOS']].sum()

    data_temp['PORC_RETRASOS'] = round((data_temp.RETRASOS/data_temp.VUELOS)*100,2)
    data_temp['PORC_VUELOS'] = round((data_temp.VUELOS/data_temp.VUELOS)*100,2)

    return data_temp

except Exception as e:
    print(f"Error al tratar de preparar los datos por hora del día. airline:{airline} {e}")
    print(e)
    print("")

```

In [141...]

```

def f_hora_prep_vis(cb_year, cb_airline):
    flights_hour_day_delay = pd.DataFrame({
        'VUELOS' : flights_df_no_cancelados.groupby(['YEAR',(flights_df_no_cancelados.SCHEDULED_DEPARTURE.dt.hour)]).sum(),
        'RETRASOS' : flights_retrasados.groupby(['YEAR',(flights_retrasados.SCHEDULED_DEPARTURE.dt.hour)]).sum()
    }).reset_index()

    flights_hour_day_delay['RETRASOS'].replace(np.nan, 0, inplace = True)
    flights_hour_day_delay.rename(columns={'SCHEDULED_DEPARTURE': 'HOUR'}, inplace=True)

    flights_hour_day_delay_porc = f_hora_agrup_porc(flights_hour_day_delay, year=cb_year)

    if flights_hour_day_delay_porc.empty:
        print("No existen datos")
    else:
        f_hora_vis(cb_year, cb_airline, flights_hour_day_delay_porc)

```

In [142...]

```

def f_hora_vis(year, airline, data):
    sns.set_color_codes("pastel")
    fig, axs = plt.subplots(nrows=2, ncols=1, figsize=(10,8), layout='constrained')

    top_bar = mpatches.Patch(color='b', label='Total vuelos')
    bottom_bar = mpatches.Patch(color='r', label='Con retraso')

    days=[]
    for n in range(0,24,1):
        days.append(n)

    graficas=2
    count_view= len(data.index)
    sns.barplot(x=data.index,y="PORC_VUELOS", data=data, color="b", ax=axs[0])
    sns_barplot(x=data.index,y="PORC_RETRASOS", data=data, color="r", ax=axs[0])
    axs[0].set_title(f'% Retrasos por hora del día a lo largo del año.\n{year} ({count_view})')
    axs[0].set_xticks(days)
    axs[0].set_xticklabels(days)
    axs[0].set(xlabel='', ylabel='')
    show_values(axs=axs[0], graficas=graficas, view_float=True, count_dec=2, text_dx=-10, text_dy=10)
    axs[0].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=(1, 0.5))

```

```

sns.barplot(x=data.index,y="VUELOS", data=data, color="b", ax=axs[1])
sns.barplot(x=data.index,y="RETRASOS", data=data, color="r", ax=axs[1])
axs[1].set_title(f'Retrasos por hora del día a lo largo del año.\n{year} {airline}')
axs[1].set_xticks(days)
axs[1].set_xticklabels(days)
axs[1].set(xlabel='', ylabel='')
show_values(axs=axs[1], graficas=graficas, view_float=False, text_rotation=90,
            axs[1].legend(handles=[top_bar, bottom_bar], loc='upper left', bbox_to_anchor=(1, 1))
plt.show()

```

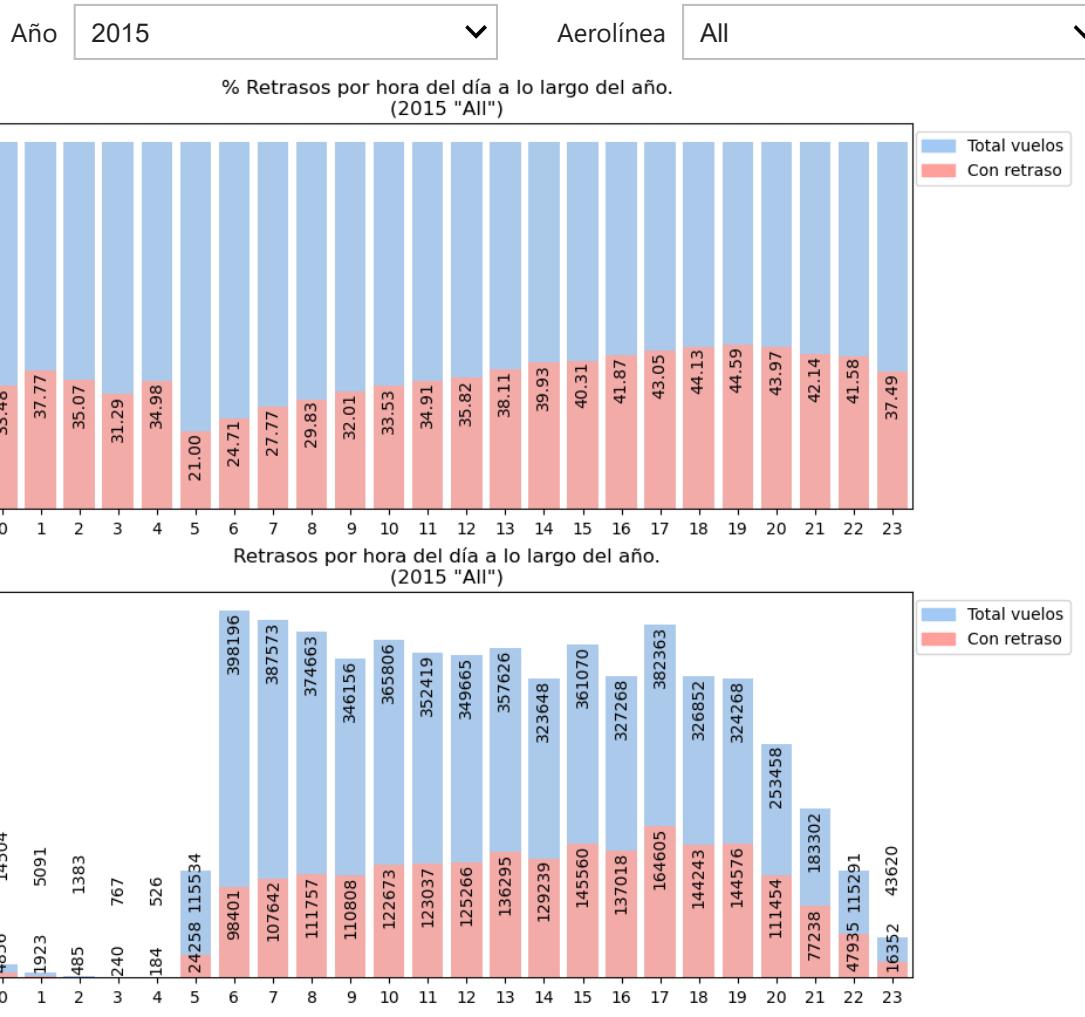
In [143...]

```

cb_airline = widgets.Dropdown(options=np.insert(flights_df_no_cancelados.AIRLINE_DICT, 0, "All"),
                               value="All", description='Aerolínea')
cb_year = widgets.Dropdown(options=['2015'], description='Año', disable=True)
ui2 = widgets.HBox([cb_year, cb_airline])

out2 = widgets.interactive_output(f_hora_prep_vis, {'cb_year':cb_year, 'cb_airline':cb_airline})
display(ui2, out2)

```



In [144...]

```

flights_hour_day_delay = pd.DataFrame()
flights_hour_day_delay_porc = pd.DataFrame()

```

Conclusión (horarios)

Los gráficos muestran claramente cómo, independientemente del mes, el número de vuelos desciende por la noche, llegando a valores muy bajos, y alcanza sus valores máximos a las 6hs, 7hs, 8hs y 10hs de la mañana, con otros dos pequeños picos a las 13hs, 15hs y 17hs.

En lo que a delay respecta, se puede notar claramente que el porcentaje comienza a aumentar a las 5hs y alcanza sus valores máximos entre 15hs y 19hs y luego vuelve a disminuir.

Es importante hacer la salvedad, que si bien el % de delay durante la madrugada es alto, no es posible arribar a conclusiones válidas dado que, tal como se mencionó previamente, el número de vuelos es ínfimo.

Conclusión

Luego de revisado el dataset y los gráficos generados, podemos concluir que si no quisieramos tener probabilidades altas de sufrir delay en el aeropuerto, tendríamos que planificar nuestro vuelo para los meses de septiembre u octubre, saliendo el día 7, 14 o 25 y, de ser posible, que alguno de los 3 caigan en sábado.

En lo que respecta al horario, es conveniente elegir que el mismo tenga horario de salida y/o llegada en las primeras horas de la mañana (5hs, 6hs o 7hs).

Analizar la cantidad de vuelos por Aerolínea

- [Volver](#)

Analizar la cantidad de vuelos por Aerolínea para determinar si alguna presenta mayor proporción de demoras respecto al resto.

```
In [145...]: DF_Proporcion = pd.DataFrame({
    'Total' : flights_df_no_cancelados.groupby('AIRLINE_DESCRIPTION').size(),
    'Retrasos' : flights_retrasados.groupby('AIRLINE_DESCRIPTION').size()
})

DF_Proporcion['Proporcion'] = round((DF_Proporcion.Retrasos/DF_Proporcion.Total)*100, 2)
DF_Proporcion['Leyenda'] = DF_Proporcion.Retrasos.astype('str') + " (" + DF_Proporcion['Proporcion'].astype('str') + "%)"

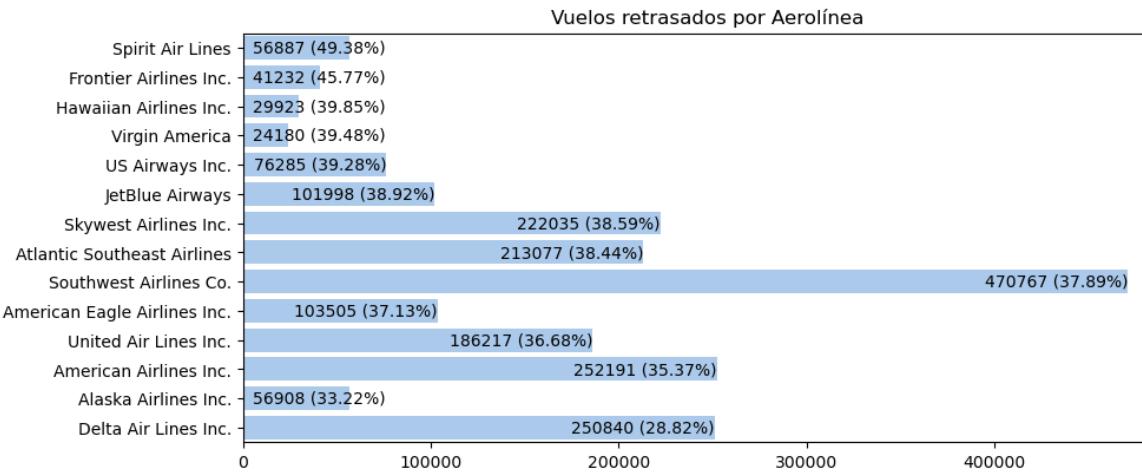
DF_Proporcion = DF_Proporcion.sort_values(by=["Proporcion", "Retrasos"], ascending=False)

In [146...]: fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(10,4), layout='constrained', dpi=100)

legends=[]
for t in DF_Proporcion['Leyenda']:
    legends.append(t)

graficas=1
count_view= len(DF_Proporcion.index)
sns.barplot(orient='h', x='Retrasos', y=DF_Proporcion.index, data=DF_Proporcion, color='red')
axs.set(xlabel='', ylabel='')
axs.set_title('Vuelos retrasados por Aerolínea', size=12)
show_values(axs=axs, orient='h', graficas=graficas, view_float=False, text_rotation=0)

plt.show()
```



In [147...]: DF_Proporcion = pd.DataFrame()

Análisis de aeropuertos de origen

- Volver
- Aeropuertos de origen con mayor cantidad de vuelos con demora por cuestiones de seguridad
- Aeropuertos de origen y sus Aerolíneas con mayor cantidad de vuelos con demora por cuestiones de seguridad.

In [148...]:

```
def format_time(time_var=None):
    """
    Desc: Función que formatea el tiempo que viene en número entero a formato HH:mm
    Ej:
        1925 -> 19:25
        557  -> 05:57
        21   -> 00:21
        2    -> 00:02

    Args:
        time_var (str): representa el número entero a formatear.

    Returns:
        str : el número formateado, si no se asigna devuelve la cadena "---:---"
    """
    if time_var==None:
        return "---:---"

    time_var=str(time_var)
    n=len(time_var)

    if n <=2:
        h=0
        m=time_var
    elif n==3:
        h = time_var[0:1]
        m = time_var[1:]
    else:
        h = time_var[0:2]
        m = time_var[2:]

    h=int(h)
```

```

m=int(m)

formattime = datetime.time(h, m, 0, 0)

return str(formattime)

```

In [149...]

```

def popup_html_top_airport_inseguros(aerolineas_data, mostrar=3, porCantidad=True):
    """
    Desc: Función que dado un DataFrame con información de vuelos Aeropuertos/Aerolíneas genera un popup html con el listado de los mismos, para las marcas en el mapa con la cantidad definida por "mostrar" (Top)

    Args:
        aerolineas_data (DataFrame): Información de Aeropuertos/Aerolíneas
        mostrar (int): Cantidad de vuelos a mostrar por Aeropuerto (Top)
        porCantidad (bool): Si muestra cantidades de demora por vuelo o el tiempo de vuelo

    Returns:
        html (str): un string con el popup html
    """

    if type(mostrar) == type(1):
        mostrar = mostrar
    else:
        mostrar=3

    left_col_color = "#19a7bd"
    right_col_color = "#f2f0d3"
    title_col_color = "#fad7a0"

    #Aeropuerto
    aiport_name = aerolineas_data['AIRPORT_DESCRIPTION'].iloc[0]
    state_name = aerolineas_data['STATE_NAME'].iloc[0]
    city = aerolineas_data['CITY'].iloc[0]

    #Aerolíneas
    airline_name = ""
    count_delay = ""
    airlines_str = ""
    airport_total_str = ""

    if porCantidad:
        total_delay = aerolineas_data['COUNT_DELAY'].sum()
        airport_total_str += \
    """
    <tr>
    <td style="width: 100px; background-color: """+ left_col_color +""";"><span style="background-color: """+ right_col_color +""";">{}</td>""".format(total_delay)
    </tr>
    """
        airlines_str += \
    """
    <tr>
    <th style="background-color: """+ title_col_color +""" align="left" colspan="2">Aerolíneas
    </th>
    """
        for i in range(0,len(aerolineas_data)):
            if i == mostrar: break
            airline_name = aerolineas_data['AIRLINE_DESCRIPTION'].iloc[i]
            count_delay = aerolineas_data['COUNT_DELAY'].iloc[i]
            airlines_str += \
    """
    <tr>
    <td style="background-color: """+ right_col_color +""";">{}</td>
    <td style="background-color: """+ left_col_color +""";">{}</td>
    </tr>
    """

```

```

<td style="width: 200px; color: #ffffff; background-color: """+ left_col_color +"""
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(count_d
</tr>
"""
    else:

        airlines_str += \
"""

<tr>
<th style="background-color: """+ title_col_color +""" align="left" colspan="4">Aerolineas
</tr>
<tr>
<td style="width: 180px; background-color: """+ title_col_color +""";"><span style="background-color: """+ title_col_color +""";">
<td style="width: 40px; background-color: """+ title_col_color +""";"><span style="background-color: """+ title_col_color +""";">
<td style="width: 110px; background-color: """+ title_col_color +""";"><span style="background-color: """+ title_col_color +""";">
<td style="width: 70px; background-color: """+ title_col_color +""";"><span style="background-color: """+ title_col_color +""";">
</tr>
"""
for i in range(0,len(aerolineas_data)):
    if i == mostrar: break
    airline_name = aerolineas_data['AIRLINE_DESCRIPTION'].iloc[i]
    date = parser.parse(str(aerolineas_data['FECHA'].iloc[i])).date()
    time = format_time(aerolineas_data['SCHEDULED_DEPARTURE'].iloc[i])
    security_delay = aerolineas_data['SECURITY_DELAY'].iloc[i]
    flight_number = aerolineas_data['FLIGHT_NUMBER'].iloc[i]
    airlines_str += \
"""

<tr>
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(airline_name)
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(flight_number)
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(date)
<td style="background-color: """+ right_col_color +""";">{} minutes.</td>""".format(security_delay)
</tr>
"""
html = """<!DOCTYPE html>
<html>
<table style="height: 20px; width: 400px;" cellspacing=0; cellpadding=0>
<tbody>
<tr>
<td style="width: 100px; background-color: """+ left_col_color +""";"><span style="background-color: """+ left_col_color +""";">
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(aiport_id)
</tr>
<tr>
<td style="width: 100px; background-color: """+ left_col_color +""";"><span style="background-color: """+ left_col_color +""";">
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(state_name)
</tr>
<tr>
<td style="width: 100px; background-color: """+ left_col_color +""";"><span style="background-color: """+ left_col_color +""";">
<td style="background-color: """+ right_col_color +""";">{}</td>""".format(city) +
</tr>
{}"".format(airport_total_str) +"""
<tr>
<table style="height: 20px; width: 400px;" cellspacing=0; cellpadding=0>
<tbody>
{}"".format(airlines_str) +"""
</tbody>
</table>
</tr>
</tbody>
</table>
"""
return html
"""

```

```
In [150...]  
mask = (flights_retrasados['SECURITY_DELAY'] > 0)  
  
top_airport_inseguros = flights_retrasados.loc[mask]  
top_airport_inseguros = pd.DataFrame({'COUNT_DELAY': top_airport_inseguros.groupby(['ORIGIN_AIRPORT']).size()})  
  
top_airport_inseguros = top_airport_inseguros.reset_index().sort_values(by=['ORIGIN_AIRPORT'], ascending=False)
```

```
In [151...]  
top_airport_inseguros = pd.merge(left=top_airport_inseguros, right=airport_data, how='left')  
top_airport_inseguros = pd.merge(left=top_airport_inseguros, right=states_data, how='left')
```

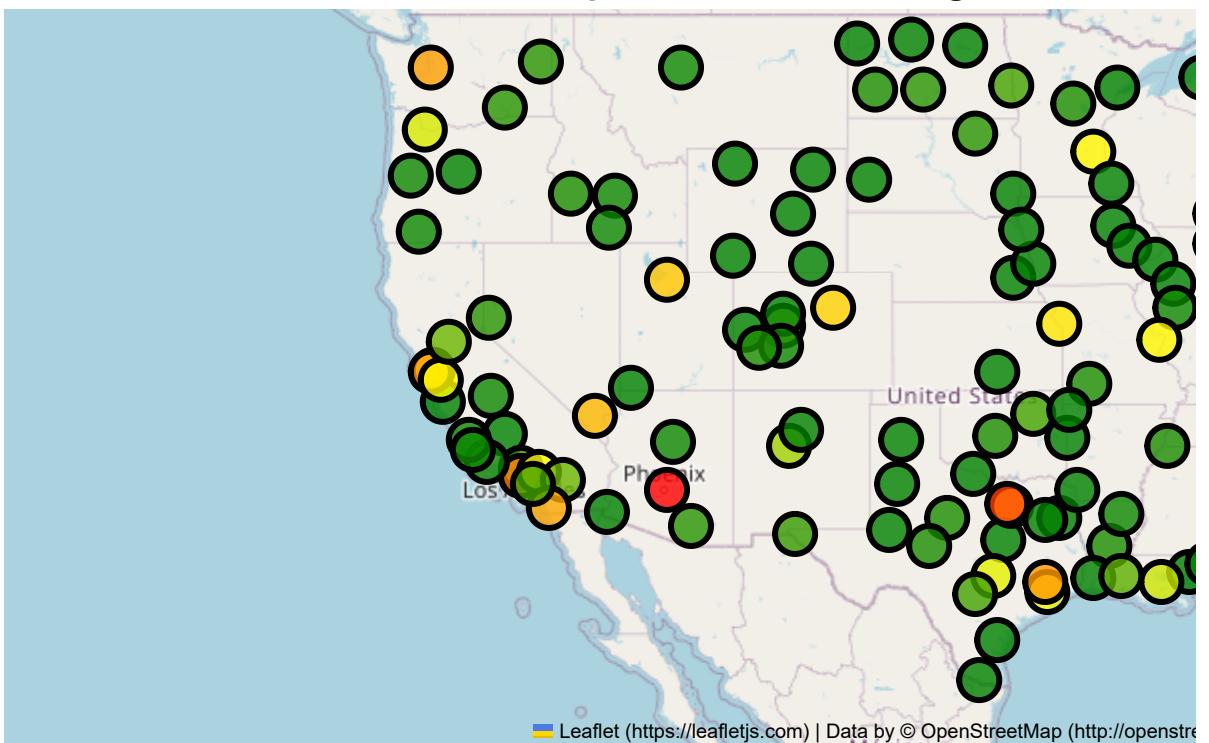
- Se prepara el mapa de colores

```
In [152...]  
mask = (flights_retrasados['SECURITY_DELAY'] > 0)  
TEMP_COUNT_DELAY = flights_retrasados[mask][['ORIGIN_AIRPORT','SECURITY_DELAY']].groupby('ORIGIN_AIRPORT').size()  
min_delay = TEMP_COUNT_DELAY.min()  
max_delay = TEMP_COUNT_DELAY.max()  
  
colormap = cm.LinearColormap(  
    ['green', 'yellow', 'orange', 'red'],  
    index=[min_delay, 20, 50, max_delay],  
    tick_labels=[1, 60, 120, 179, 238, 298, 357],  
    vmin=min_delay,  
    vmax=max_delay)  
#colormap
```

```
In [153...]  
location = top_airport_inseguros['LATITUDE'].mean(), top_airport_inseguros['LONGITUDE'].mean()  
  
usa_map = folium.Map(width="100%", height="92%", location=location, zoom_start=4, max_zoom=10)  
usa_map.fit_bounds([-125, 25, -60, 50])  
max_bounds = True, min_lat=12, max_lat=73, min_lon=-178, max_lon=-60  
  
colormap.height = 40  
colormap.width = 200  
colormap.caption = "Cantidad de vuelos con demora"  
colormap.add_to(usa_map)  
  
for aport in top_airport_inseguros['ORIGIN_AIRPORT'].unique():  
  
    mask = (top_airport_inseguros['ORIGIN_AIRPORT']==aport)  
    top_aerolineas = top_airport_inseguros[mask].copy(deep=True)  
  
    html = popup_html_top_airport_inseguros(top_aerolineas, 10, True)  
  
    iframe = folium.IFrame(html=html, width=440)  
  
    popup = folium.Popup(folium.Html(html, script=True), max_width=430)  
  
    folium.CircleMarker([top_aerolineas['LATITUDE'].iloc[0], top_aerolineas['LONGITUDE'].iloc[0]],  
        radius=10,  
        popup=popup,  
        color='black',  
        fill_color=colormap(TEMP_COUNT_DELAY[aport]),  
        fill_opacity=0.8  
    ).add_to(usa_map)  
  
if isinstance(usa_map, folium.Map):  
    figure = folium.Figure(width = 900, height = 400).add_child(usa_map)  
  
#Título  
title_html = '''<div align="center" style="font-size:18px; color:black;">  
    <b>Cantidad total de demoras por cuestiones de seguridad en Aeropuertos/Aerolíneas</b>
```

```
figure.get_root().html.add_child(folium.Element(title_html));  
  
usa_map
```

Out[153]:



```
In [154...]: TEMP_COUNT_DELAY = pd.DataFrame()
          top_aerolineas = pd.DataFrame()
          top_airport_inseguros = pd.DataFrame()
```

- Aeropuertos de origen y sus Aerolíneas con vuelos de mayor demora por cuestiones de seguridad.

```
In [155...]: top_airport_inseguros_delay = flights_retrasados.sort_values(by=['SECURITY_DELAY'])  
top_airport_inseguros_delay = pd.merge(left=top_airport_inseguros_delay, right=
```

```
In [156]: location = top_airport_inseguros_delay['LATITUDE'].mean(), top_airport_inseguros_delay['LONGITUDE'].mean()
usa_map = folium.Map(width="100%", height="92%", location=location, zoom_start=4, max_bounds=True, min_lat=12, max_lat=73, min_lon=-178, max_lon=-60)
for aport in top_airport_inseguros_delay['ORIGIN_AIRPORT'].unique():
    mask = (top_airport_inseguros_delay['ORIGIN_AIRPORT'] == aport)
    top_aerolineas_delay = top_airport_inseguros_delay[mask].copy(deep=True)
    html = popup_html_top_airport_inseguros(top_aerolineas_delay, 25, False)
    iframe = folium.IFrame(html=html, width=440)
    popup = folium.Popup(folium.Html(html, script=True), max_width=430)
    folium.CircleMarker([top_aerolineas_delay['LATITUDE'].iloc[0], top_aerolineas_delay['LONGITUDE'].iloc[0]], radius=top_airport_inseguros_delay[mask]['SECURITY_DELAY'].max(), popup=popup, color='blue', fill_color='blue', fill=True).add_to(usa_map)
```

```

        fill_opacity=0.3).add_to(usa_map)

if isinstance(usa_map, folium.Map):
    figure = folium.Figure(width = 900, height = 450 ).add_child( usa_map)

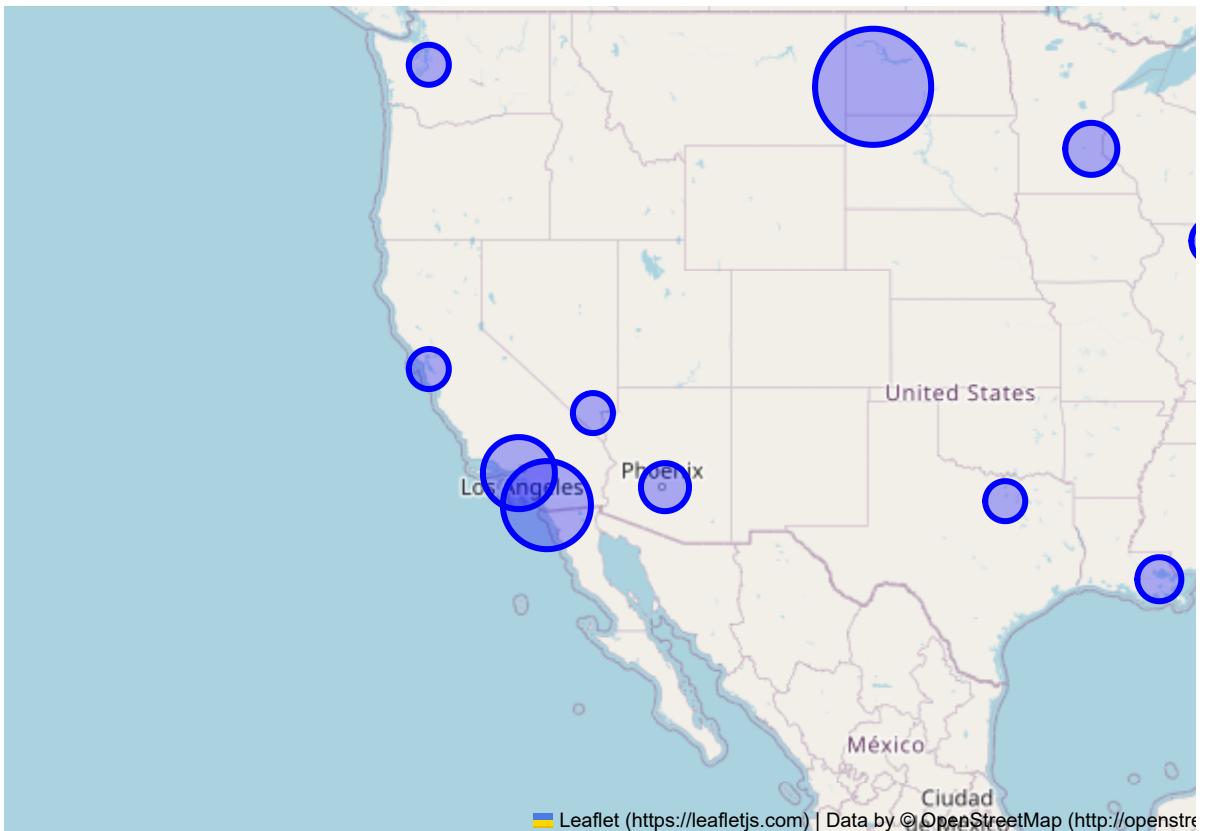
#Título
title_html ='''<div align="center" style="font-size:18px; color:black;">
<b>Aerolíneas con vuelos de mayor demora por cuestiones de seguridad (Top 25)</b></div>'''
figure.get_root().html.add_child(folium.Element(title_html));

usa_map

```

Out[156]:

Aerolíneas con vuelos de mayor demora por cuestiones



In [157...]

```

top_aerolineas_delay = pd.DataFrame()
top_airport_inseguros_delay = pd.DataFrame()

```

Análisis (Hip.1) e (Hip.2)

- Volver
- Las demoras en la mayoría de los vuelos se dan a causa de factores climáticos.
- La disminución de la cantidad y tiempo de demora en los vuelos puede lograrse si se disminuyen los controles de seguridad en los aeropuertos.

In [158...]

```

# Se crea el DF de frecuencias acumuladas
df_freq_causas_delay = pd.DataFrame(columns=['n', 'frecuencia', 'porcentaje', 'porc_acumulado'])
df_freq_causas_delay['frecuencia'] = [1, 2, 3, 4, 5]
# Se genera el Index
df_freq_causas_delay.index = ['AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY']

```

```

# Se genera un indice "n" de la columna 0
df_freq_causas_delay['n'] = 1
df_freq_causas_delay['n'] = df_freq_causas_delay['n'].cumsum()

df_freq_causas_delay = df_freq_causas_delay.fillna(0)

# Se calculan las frecuencias
mask = (flights_retrasados['AIR_SYSTEM_DELAY'] > 0)
frecuencia = len(flights_retrasados.loc[mask])
df_freq_causas_delay.loc['AIR_SYSTEM_DELAY', 'frecuencia'] = frecuencia

mask = (flights_retrasados['SECURITY_DELAY'] > 0)
frecuencia = len(flights_retrasados.loc[mask])
df_freq_causas_delay.loc['SECURITY_DELAY', 'frecuencia'] = frecuencia

mask = (flights_retrasados['AIRLINE_DELAY'] > 0)
frecuencia = len(flights_retrasados.loc[mask])
df_freq_causas_delay.loc['AIRLINE_DELAY', 'frecuencia'] = frecuencia

mask = (flights_retrasados['LATE_AIRCRAFT_DELAY'] > 0)
frecuencia = len(flights_retrasados.loc[mask])
df_freq_causas_delay.loc['LATE_AIRCRAFT_DELAY', 'frecuencia'] = frecuencia

mask = (flights_retrasados['WEATHER_DELAY'] > 0)
frecuencia = len(flights_retrasados.loc[mask])
df_freq_causas_delay.loc['WEATHER_DELAY', 'frecuencia'] = frecuencia

# Se ordena el DF
df_freq_causas_delay = df_freq_causas_delay.sort_values('frecuencia', ascending=False)

# Se reinicia el indice "n" de la columna 0
df_freq_causas_delay['n'] = 1
df_freq_causas_delay['n'] = df_freq_causas_delay['n'].cumsum()

df_freq_causas_delay['porcentaje'] = df_freq_causas_delay['frecuencia'] / df_freq_causas_delay['frecuencia'].sum()

i=0
while i < len(df_freq_causas_delay):
    if i == 0:
        df_freq_causas_delay.iloc[i, 3] = df_freq_causas_delay['porcentaje'][i]
    else:
        df_freq_causas_delay.iloc[i, 3] = (df_freq_causas_delay['porcentaje'][i] + df_freq_causas_delay['porcentaje'][i-1])
    i += 1

```

In [159...]

```

values = np.array(df_freq_causas_delay['frecuencia'])
values_sorted = values
values_acum = np.array(df_freq_causas_delay['porcentaje_acum'])

n = len(values)

```

In [160...]

```

import numpy as np
import matplotlib.pyplot as plt

fig, axs = plt.subplots(nrows=1, ncols=1, figsize=(8,5), layout='constrained')

sns.barplot(x=df_freq_causas_delay.index, y=values, color="b", ax=axs, width=0.5,
            xlabel='Causas de retrasos', ylabel='Cantidad de ocurrencias', ylim=(0, 60))

plt.xticks(rotation=45)

ax2 = axs.twinx()
sns.lineplot(x=df_freq_causas_delay.index, y=values_acum, color="C1", marker='o', 

```

```

dict_prop={"facecolor":'black', "shrink":0.05}
axs.annotate(str(round(values_acum[2]*100,2))+"%", xy=(2.05, 514000), xytext=(2.3,
                                         arrowprops=dict_prop)

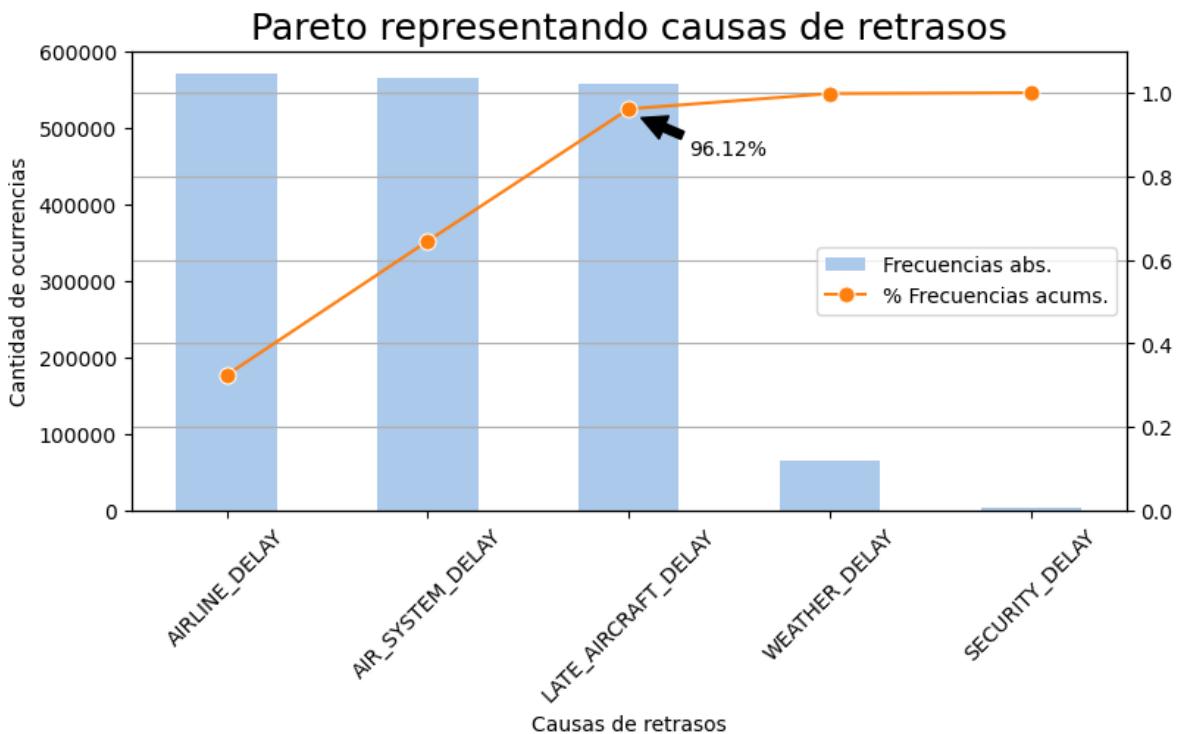
plt.title("Pareto representando causas de retrasos", fontdict={'size':18})

lines, labels = axs.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc="center right")

# axis: x0,x1,y0, y1
plt.axis(ymin=0, ymax=1.1)
plt.grid()

plt.show()

```



Conclusión

El "Principio de Pareto" expresa que existe una relación 20% - 80% entre las causas de un problema y los efectos de los mismos. Es decir, el 80% del efecto de un problema, se debe solo al 20% de las causas.

Basado en los datos y este Principio, se logra identificar las causas esenciales que originan los retrasos en los vuelos.

El 60% (3 de 5) de las causas de las demoras en los vuelos radica en retrasos originados en las Aerolíneas, en el Sistema de Control de Tráfico Aéreo, y en el retraso de vuelos previos de la aeronave, los cuales generan el 96% de los retrasos.

Se concluye que las Hipótesis planteadas no se corroboran por los datos, y son Falsas.

Análisis (Hip.3)

- [Volver](#)
- Se analiza que el tiempo de demora de salida y llegada de los vuelos son, en su mayoría, positivos y altos.

Promedios

- [Volver](#)

Mediana y Moda

```
In [161...]:  
values_departure_delay = np.array(flights_df_no_cancelados['DEPARTURE_DELAY'])  
txt_departure_delay = 'DEPARTURE_DELAY'  
  
values_arrival_delay = np.array(flights_df_no_cancelados['ARRIVAL_DELAY'])  
txt_arrival_delay = 'ARRIVAL_DELAY'  
  
values_taxi_out = np.array(flights_df_no_cancelados['TAXI_OUT'])  
txt_taxi_out = 'TAXI_OUT'  
  
values_taxi_in = np.array(flights_df_no_cancelados['TAXI_IN'])  
txt_taxi_in = 'TAXI_IN'  
  
In [162...]:  
sorted_departure_delay = np.sort(values_departure_delay),  
mediana_departure_delay = np.median(sorted_departure_delay)  
  
sorted_arrival_delay = np.sort(values_arrival_delay),  
mediana_arrival_delay = np.median(sorted_arrival_delay)  
  
sorted_taxi_out = np.sort(values_taxi_out),  
mediana_taxi_out = np.median(sorted_taxi_out)  
  
sorted_taxi_in = np.sort(values_taxi_in),  
mediana_taxi_in = np.median(sorted_taxi_in)  
  
moda_departure_delay, count_departure_delay = stats.mode(values_departure_delay)  
moda_arrival_delay, count_arrival_delay = stats.mode(values_arrival_delay)  
moda_taxi_out, count_taxi_out = stats.mode(values_taxi_out)  
moda_taxi_in, count_taxi_in = stats.mode(values_taxi_in)  
  
print(f'{txt_departure_delay}')  
print(f'La Mediana es: {mediana_departure_delay}')  
print(f'La Moda es: {moda_departure_delay}. La frecuencia es: {count_departure_delay}')  
  
print(f'{txt_arrival_delay}')  
print(f'La Mediana es: {mediana_arrival_delay}')  
print(f'La Moda es: {moda_arrival_delay}. La frecuencia es: {count_arrival_delay}')  
  
print(f'{txt_taxi_out}')  
print(f'La Mediana es: {mediana_taxi_out}')  
print(f'La Moda es: {moda_taxi_out}. La frecuencia es: {count_taxi_out}\n\n')  
  
print(f'{txt_taxi_in}')  
print(f'La Mediana es: {mediana_taxi_in}')  
print(f'La Moda es: {moda_taxi_in}. La frecuencia es: {count_taxi_in}\n\n')
```

```
DEPARTURE_DELAY  
La Mediana es: -2.0  
La Moda es: [-3]. La frecuencia es: [454245]
```

```
ARRIVAL_DELAY  
La Mediana es: -5.0  
La Moda es: [-8]. La frecuencia es: [176779]
```

```
TAXI_OUT  
La Mediana es: 14.0  
La Moda es: [12]. La frecuencia es: [461843]
```

```
TAXI_IN  
La Mediana es: 6.0  
La Moda es: [5]. La frecuencia es: [930526]
```

Medidas de Dispersión

- [Volver](#)

Media y Desviación Estándar

```
In [163...]  
mu_departure_delay = np.mean(values_departure_delay)  
mu_arrival_delay = np.mean(values_arrival_delay)  
mu_taxi_out = np.mean(values_taxi_out)  
mu_taxi_in = np.mean(values_taxi_in)
```

```
In [164...]  
sigma_departure_delay = np.std(values_departure_delay)  
sigma_arrival_delay = np.std(values_arrival_delay)  
sigma_taxi_out = np.std(values_taxi_out)  
sigma_taxi_in = np.std(values_taxi_in)  
  
print (f'La Media de {txt_departure_delay} es: {mu_departure_delay}')  
print (f'El Desvío Estándar de {txt_departure_delay} es: {sigma_departure_delay}\n')  
  
print (f'La Media de {txt_arrival_delay} es: {mu_arrival_delay}')  
print (f'El Desvío Estándar de {txt_arrival_delay} es: {sigma_arrival_delay}\n')  
  
print (f'La Media de {txt_taxi_out} es: {mu_taxi_out}')  
print (f'El Desvío Estándar de {txt_taxi_out} es: {sigma_taxi_out}\n')  
  
print (f'La Media de {txt_taxi_in} es: {mu_taxi_in}')  
print (f'El Desvío Estándar de {txt_taxi_in} es: {sigma_taxi_in}\n')
```

```

La Media de DEPARTURE_DELAY es: 9.299625165184189
El Desvío Estándar de DEPARTURE_DELAY es: 36.892886359278805

La Media de ARRIVAL_DELAY es: 4.4098763642196035
El Desvío Estándar de ARRIVAL_DELAY es: 39.27511438036711

La Media de TAXI_OUT es: 16.066589167769354
El Desvío Estándar de TAXI_OUT es: 8.883081785342908

La Media de TAXI_IN es: 7.429900531408503
El Desvío Estándar de TAXI_IN es: 5.6198956325662985

```

In [165...]

```

x_ax0 = values_departure_delay
x_ax1 = values_arrival_delay
x_ax2 = values_taxi_out
x_ax3 = values_taxi_in

fig, ((ax0, ax2), (ax1, ax3)) = plt.subplots(nrows=2, ncols=2, figsize=(14,12))
fig.subplots_adjust(left=0.1, right=0.95, bottom=0.3)

# ax0
xmin_ax0= -100
xmax_ax0= 150

intervalos = []
i = xmin_ax0
while i <= xmax_ax0:
    intervalos.append(i)
    i += 5

ax0.hist(x_ax0, bins=intervalos, density=True, edgecolor = 'black', linewidth=0.5)
ax0.set_title(f"Histograma y curva Normal de \nMinutos de Retraso de Salida", fontdict={'size':10})
ax0.set_xlabel("Variación en Minutos (5 min)", fontdict={'size':10})
ax0.set_ylabel("Cantidad de ocurrencias respecto del Total", fontdict={'size':10})
ax0.grid(True)

#ax1
xmin_ax1= -100
xmax_ax1= 150

intervalos = []
i = xmin_ax1
while i <= xmax_ax1:
    intervalos.append(i)
    i += 5

ax1.hist(x_ax1, intervalos, density=True, edgecolor = 'black', linewidth=0.5)
ax1.set_title(f"Histograma y curva Normal de \nMinutos de Retraso de Llegada", fontdict={'size':10})
ax1.set_xlabel("Variación en Minutos (5 min)", fontdict={'size':10})
ax1.set_ylabel("Cantidad de ocurrencias respecto del Total", fontdict={'size':10})
ax1.grid(True)

#ax2
xmin_ax2= -50
xmax_ax2= 100

intervalos = []
i = xmin_ax2
while i <= xmax_ax2:
    intervalos.append(i)
    i += 5

ax2.hist(x_ax2, intervalos, density=True, edgecolor = 'black', linewidth=0.5)

```

```

ax2.set_title(f"Histograma y curva Normal de \nMinutos de Retraso de Salida en la Pista", fontdict={'size':10})
ax2.set_xlabel("Variación en Minutos (5 min)", fontdict={'size':10})
ax2.set_ylabel("Cantidad de ocurrencias respecto del Total", fontdict={'size':10})
ax2.grid(True)

#ax3
xmin_ax3= -50
xmax_ax3= 100

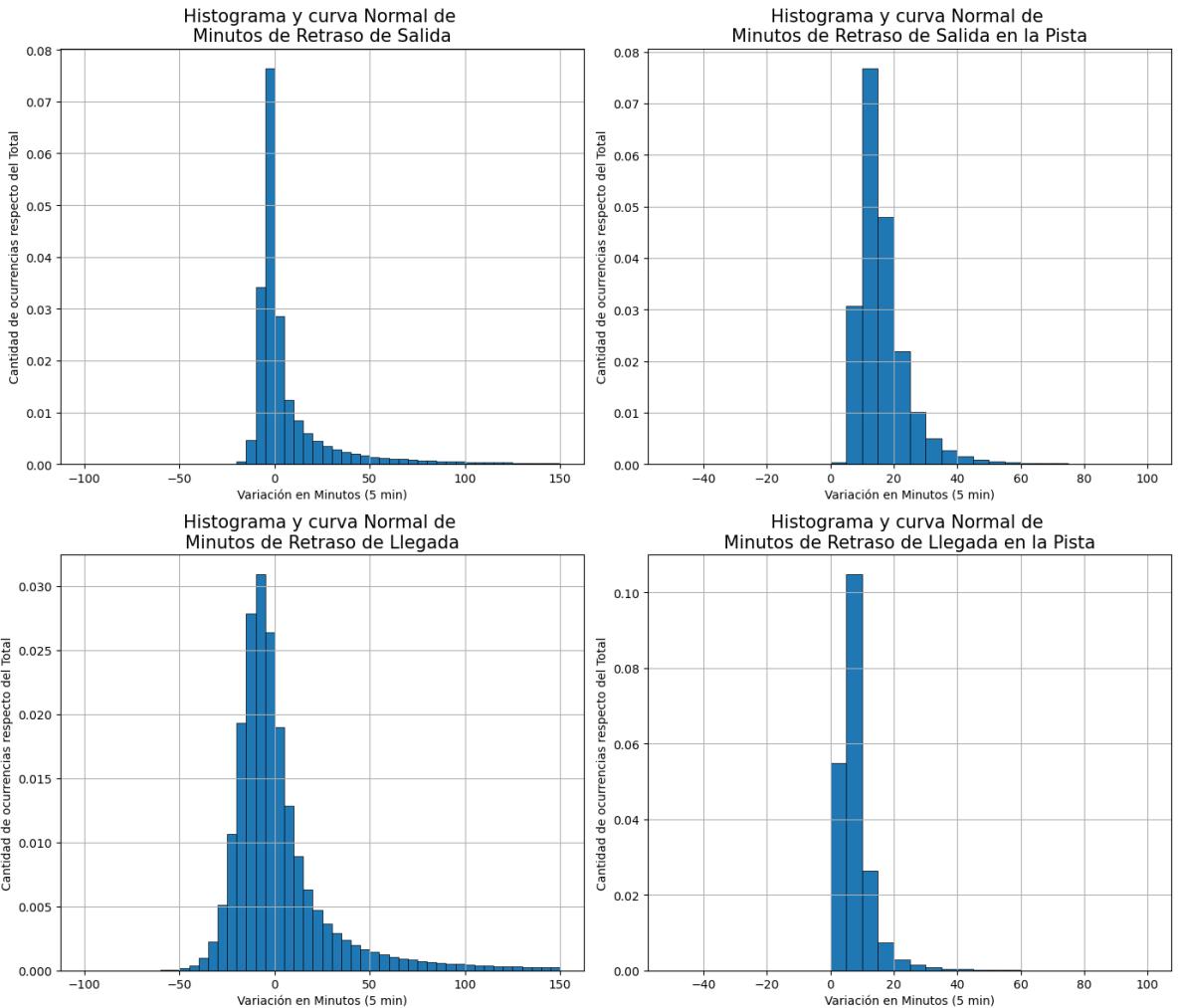
intervalos = []
i = xmin_ax3
while i <= xmax_ax3:
    intervalos.append(i)
    i += 5

ax3.hist(x_ax3, intervalos, density=True, edgecolor = 'black', linewidth=0.5)
ax3.set_title(f"Histograma y curva Normal de \nMinutos de Retraso de Llegada en la Pista", fontdict={'size':10})
ax3.set_xlabel("Variación en Minutos (5 min)", fontdict={'size':10})
ax3.set_ylabel("Cantidad de ocurrencias respecto del Total", fontdict={'size':10})
ax3.grid(True)

fig.tight_layout()

plt.show()

```



Conclusión

Analizados los datos y la cantidad de ocurrencias de distintos períodos de tiempo de demora (positivos y negativos), puede observarse lo siguiente:

- Minutos de Retraso de Salida: La mayor cantidad de ocurrencias se presentan en los 5 minutos previos al horario programado de salida (partida temprana del vuelo).
- Minutos de Retraso de Llegada: La mayor cantidad de ocurrencias se presentan en los 10 minutos previos al horario programado de llegada (llegada temprana del vuelo).

Se concluye que la Hipótesis planteada no se corrobora por los datos, y es Falsa.

Nota: Para contabilizar correctamente el retraso de salida, se debería conocer el taxi-out planificado y el taxi-in planificado para sumar al departure-delay el retardo asociado del taxi-out y tener un retraso de salida más preciso, dado que no se puede inferir solo con las columnas de delay, cuanto tiempo de retraso se distribuye entre taxi-out, taxi-in, o air-time.

Almacenamiento de resultados

- [Volver](#)
- Se exporta a CSV

In [166...]

```
flights_retrasados.to_csv(data_path + data_result + 'flights_retrasados.csv', index=False)
flights_df_no_cancelados.to_csv(data_path + data_result + 'flights_df_no_cancelados.csv', index=False)

print(f'\nSe crearon dos archivos en formato CSV con la información solicitada.\n')
```

Se crearon dos archivos en formato CSV con la información solicitada.

El nombre de los archivos son: "flights_retrasados.csv" y "flights_df_no_cancelados.csv".

Se encuentran en "./data_result/" del presente notebook.